DYNAMIC OPPONENT MODELLING IN TWO-PLAYER GAMES

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2015

By
Richard Andrew Mealing
School of Computer Science

Contents

List of Abbreviations **11**

Abstract **13**

Declaration **14**

Copyright **15**

Publications **16**

Acknowledgements **17**

1 Introduction **18**

1.1 Why Study Multi-Agent Learning? 18

1.2 Why Study Games? 20

1.3 Learning to Act 22

1.4 What is this thesis about? 23

1.4.1 Scope 25

1.4.2 Contributions of this Thesis 25

1.4.3 Thesis Structure 28

2 Problem Definition **32**

2.1 How Should an Agent Act? 32

2.2 Learning Objectives 35

2.3 Assumptions 37

2.4 Chapter Summary 40

3 Background and Related Work **41**

3.1 Game Theory 41

3.1.1 What is a Game? 42
3.1.2 Game Classes 44
3.1.3 Game Categorisations 47
3.1.4 Strategy Categorisations 55
3.1.5 Solution Concepts 56
3.1.6 Solving Games 62
3.1.7 Games in the Experiments 65
3.1.8 Abstraction by Bucketing 73
3.2 Learning in Games 75
3.2.1 Repeating Games 75
3.2.2 Stochasticity in Games 76
3.2.3 The Problem of Convergence 76
3.2.4 Modes of Convergence 77
3.2.5 Convergence in Game Theory 79
3.2.6 Machine Learning 81
3.2.7 Reinforcement Learning 83
3.2.8 Reinforcement Learning Algorithms 84
3.2.9 Counterfactual Regret Minimisation 93
3.2.10 Opponent Modelling 99
3.2.11 Value Estimation by Lookahead 110
3.3 Chapter Summary 112

4 OM with SP, RL, and Lookahead **113**
4.1 Sequence Prediction Opponent Modelling 114
4.1.1 Short-Term and Long-Term Memory 115
4.1.2 Sequence Prediction Methods 115
4.2 Lookahead 116
4.3 Games in the Experiments 117
4.4 Opponents in the Experiments 117
4.5 The Approach 119
4.6 Comparison to Reinforcement Learning Agents 120
4.6.1 Iterated Rock-Paper-Scissors 124
4.6.2 Iterated Prisoner's Dilemma 126
4.6.3 Littman's Soccer Game 130
4.7 Chapter Summary 131

5 OM with EM, SP, and No-Regret Learning 133
5.1 Opponent Modelling in Poker . . . 136
5.2 Expectation-Maximisation . . . 139
5.3 Games in the Experiments . . . 140
5.4 Opponents in the Experiments . . . 141
5.5 The Approach . . . 142
5.5.1 Expectation-Maximisation in the Opponent Model . . . 142
5.5.2 Sequence Prediction in the Opponent Model . . . 147
5.5.3 Operation and Algorithm . . . 150
5.6 Evaluating the Three Proposals . . . 154
5.6.1 Benefit of bucketed Rhode Island Hold'em . . . 155
5.6.2 Performance in Die-Roll Poker and Rhode Island Hold'em against the Opponents . . . 157
5.6.3 Further Experiments in Rhode Island hold'em . . . 164
5.7 Chapter Summary . . . 170

6 Change Detection for Opponent Strategies 171
6.1 Learning a Changing Distribution . . . 173
6.2 Adapting the Window Size . . . 173
6.2.1 ADaptive WINdowing (ADWIN) . . . 174
6.2.2 Bayesian Change Point Detection (BayesCPD) . . . 179
6.2.3 Fisher's Exact Test Change Point Model (FET-CPM) . . . 181
6.3 Comparing Change Detection Methods . . . 183
6.3.1 Modelling Changing Categorical Distributions . . . 185
6.3.2 Modelling Reinforcement Learning Agents . . . 195
6.4 Combining SP and CD . . . 199
6.5 Chapter Summary . . . 200

7 Dynamic OM Self-Play Convergence 202
7.1 Fictitious Play Example . . . 203
7.2 Extensions to Fictitious Play . . . 206
7.3 SP and CD Self-Play Convergence . . . 207
7.3.1 Combining Sequence Prediction and Fictitious Play . . . 212
7.3.2 Combining Change Detection and Fictitious Play . . . 212
7.4 Empirical Results for Self-Play Convergence . . . 213
7.4.1 Measuring Nash Equilibrium Convergence . . . 214

7.4.2 Maximum Convergence Rate 215
7.4.3 Matching Pennies 220
7.4.4 Generalised Matching Pennies 222
7.4.5 Shapley's Game 226
7.4.6 Jordan's Game 229
7.5 Chapter Summary 231

8 Conclusions and Future Work 233
8.1 Conclusions 233
8.1.1 First Approach: Sequence Prediction, Reinforcement Learning, and Lookahead 234
8.1.2 Second Approach: Expectation-Maximisation, Sequence Prediction, and No-Regret Learning 234
8.1.3 Sequence Prediction Methods 235
8.1.4 Change Detection Methods 236
8.1.5 Self-Play Convergence of Empirical Distributions of Plays . 236
8.2 Future Work 237
8.2.1 Safe Strategy 237
8.2.2 Opponent Strategy Space 238
8.2.3 Sequence Prediction Methods 238
8.2.4 Continuous Values 239
8.2.5 N-player games 239
8.2.6 Learning Objectives 240
8.2.7 Real-world Problems 241
8.2.8 Prior Knowledge 241
8.2.9 Implicit Modelling 241
8.2.10 Sequence Prediction and Change Detection 242

Bibliography 243

Appendices 252

A Expectation-Maximisation 253
A.1 Categorical Distribution 253
A.2 Mixture of Categorical Distributions 255

B 2-Player 2-Action Nash Equilibrium Payoffs 259

B.1 Payoffs for a Nash Equilibrium at a Centre Point 260

B.2 Payoffs for a Nash Equilibrium at a Saddle Point 261

C Simplex Sample and Simplex Random Walk 264

C.1 $K-1$-Dimensional Simplex Sample 264

C.2 $K-1$-Dimensional Simplex Random Walk 267

C.2.1 Metropolis-Hastings Algorithm 268

C.2.2 Application of the Metropolis-Hastings Algorithm 268

D Fictitious Play Example Details 270

Word Count: 61886

List of Tables

1.1 Artificial intelligence success in games. 21

3.1 Terminology comparison. 43
3.2 Rock-paper-scissors payoff matrix. 65
3.3 Shapley's game payoff matrix. 66
3.4 Miscoordination game payoff matrix. 66
3.5 Matching pennies payoff matrix. 67
3.6 Jordan's game payoff matrix. 67
3.7 Prisoner's dilemma payoff matrix. 68
3.8 Three-card poker hand ranks. 71
3.9 Hierarchical n-gram matching pennies example. 105

4.1 Iterated prisoner's dilemma opponents. 118
4.2 Sequence prediction opponent modelling experiment parameters. . 123
4.3 Iterated rock-paper-scissors results. 125
4.4 Iterated prisoner's dilemma results. 127
4.5 Littman's soccer game results. 131

5.1 Parameters used in the experiments. 155
5.2 Die-roll poker modelled average payoffs per game. 161
5.3 Rhode Island hold'em modelled average payoffs per game. 163
5.4 Rhode Island hold'em opponent hidden information dependence. . 169
5.5 Final average payoff per game of (Expectation-Maximisation (EM) + SP) - (SP) in die-roll poker. 169

6.1 Data from two Bernoulli samples. 182
6.2 Methods vs changing distributions parameters. 187
6.3 Example single run change detection results for 3 categories. . . . 193
6.4 Average change detection results. 193

6.5 Methods vs reinforcement learning agents parameters. 196
6.6 Matching pennies average distances. 198
6.7 Matching pennies average correct predictions. 198
6.8 Matching pennies average rewards. 198
6.9 Rock-paper-scissors average distances. 199
6.10 Rock-paper-scissors average correct predictions. 199
6.11 Rock-paper-scissors average rewards. 199

7.1 Generalised matching pennies mean convergence rate and distance estimates. 226

B.1 2x2 centre Nash equilibrium payoffs. 260
B.2 2x2 saddle Nash equilibrium payoffs. 261

List of Figures

3.1 An arbitrary game in extensive-form, normal-form, and sequence-form. 54
3.2 Littman's soccer game. 69
3.3 Die-roll poker game tree. 70
3.4 Rhode Island hold'em poker game tree. 72

5.1 Expectation-maximisation sequence prediction flowchart. 152
5.2 Final average payoffs per game in abstracted and unabstracted Rhode Island hold'em. 156
5.3 Final average payoffs per game of OS-MCCFR with opponent modelling. 160
5.4 Rhode Island hold'em Additional Results. 166

6.1 Change detection single run results. 192

7.1 Matching pennies expected dynamics of fictitious play in self-play. 205
7.2 Matching pennies recurrence plots. 211
7.3 Matching pennies sequence prediction and change detection in self-play. 222
7.4 Generalised matching pennies various methods in self-play. 225
7.5 Shapley's game various methods in self-play. 228
7.6 Jordan's game various methods in self-play. 230

B.1 Gradients around a centre Nash equilibrium at (p* = 0.5, q* = 0.5).261
B.2 Gradients around a saddle Nash equilibrium at (p* = 0.5, q* = 0.5).263

List of Algorithms

1 Q(uality)-Learning 86
2 Win or Learn Fast Policy Hill Climbing (WoLF-PHC) 88
3 Weighted Policy Learner (WPL) 90
4 Policy Gradient Ascent with Approximate Policy Prediction (PGA-APP) 92
5 A General Sequence Prediction Method 107
6 Sequence Prediction with Q-Learning and Lookahead 120
7 Entropy Learned Pruned Hypothesis Space (ELPH) 149
8 Opponent Model using Sequence Prediction and EM 153
9 Estimating Strategy Dependence on Hidden Information 168
10 ADaptive WINdowing (ADWIN) 175
11 ADaptive WINdowing using Distance (ADWIN-D) 176
12 ADaptive WINdowing using Bernoulli distributions (ADWIN-B) . 178
13 Stratified Optimal Resampling (SOR) 181
14 Fisher's Exact Test Change Point Model using Bernoulli distributions (FET-CPM-B) 183
15 Metropolis-Hastings Unit-Exponential Sample 186
16 Sequence Prediction and Fictitious Play 212
17 Bayes Model Modified Probability 213

List of Abbreviations

ADWIN ADaptive WINdowing.

ADWIN-B ADaptive WINdowing using Bernoulli distributions.

ADWIN-D ADaptive WINdowing using Distance.

BayesCPD Bayesian Change Point Detection.

BayesCPD-B Bayesian Change Point Detection using Bernoulli distributions.

BayesCPD-C Bayesian Change Point Detection using Categorical distributions.

CD Change Detection.

CDFP Change Detection and Fictitious Play.

CFR Counterfactual Regret Minimisation.

CFRX Counterfactual Regret Minimisation iterated X times.

CUSUM CUMulative SUM Control Chart.

ELPH Entropy Learned Pruned Hypothesis Space.

EM Expectation-Maximisation.

FET-CPM Fisher's Exact Test Change Point Model.

FET-CPM-B Fisher's Exact Test Change Point Model using Bernoulli distributions.

FP Fictitious Play.

KMP Knuth-Morris-Pratt.

LSTM Long Short-Term Memory.

LZ78 Lempel-Ziv-1978.

MCCFR Monte Carlo Counterfactual Regret Minimisation.

MDP Markov Decision Process.

OS-MCCFR Outcome Sampling Monte Carlo Counterfactual Regret Minimisation.

OS-MCCFR OM Outcome Sampling Monte Carlo Counterfactual Regret Minimisation with an Opponent Model.

PGA-APP Policy Gradient Ascent with Approximate Policy Prediction.

PPMC Prediction by Partial Matching version C.

Q-Learning Quality-Learning.

SEM Standard Error of the Mean.

SOR Stratified Optimal Resampling.

SP Sequence Prediction.

SP-OM Sequence Prediction Opponent Modelling.

SPFP Sequence Prediction and Fictitious Play.

TDAG Transition Directed Acyclic Graph.

UCB Upper Confidence Bounds.

WoLF Win or Learn Fast.

WoLF-IGA Win or Learn Fast Iterated Gradient Ascent.

WoLF-PHC Win or Learn Fast Policy Hill Climbing.

WPL Weighted Policy Learner.

Abstract

Dynamic Opponent Modelling in Two-Player Games

Richard Andrew Mealing

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy, 2015

This thesis investigates decision-making in two-player imperfect information games against opponents whose actions can affect our rewards, and whose strategies may be based on memories of interaction, or may be changing, or both. The focus is on modelling these dynamic opponents, and using the models to learn high-reward strategies. The main contributions of this work are: 1. An approach to learn high-reward strategies in small simultaneous-move games against these opponents. This is done by using a model of the opponent learnt from sequence prediction, with (possibly discounted) rewards learnt from reinforcement learning, to lookahead using explicit tree search. Empirical results show that this gains higher average rewards per game than state-of-the-art reinforcement learning agents in three simultaneous-move games. They also show that several sequence prediction methods model these opponents effectively, supporting the idea of using them from areas such as data compression and string matching; 2. An online expectation-maximisation algorithm that infers an agent's hidden information based on its behaviour in imperfect information games; 3. An approach to learn high-reward strategies in medium-size sequential-move poker games against these opponents. This is done by using a model of the opponent learnt from sequence prediction, which needs its hidden information (inferred by the online expectation-maximisation algorithm), to train a state-of-the-art no-regret learning algorithm by simulating games between the algorithm and the model. Empirical results show that this improves the no-regret learning algorithm's rewards when playing against popular and state-of-the-art algorithms in two simplified poker games; 4. Demonstrating that several change detection methods can effectively model changing categorical distributions with experimental results comparing their accuracies to empirical distributions. These results also show that their models can be used to outperform state-of-the-art reinforcement learning agents in two simultaneous-move games. This supports the idea of modelling changing opponent strategies with change detection methods; 5. Experimental results for the self-play convergence to mixed strategy Nash equilibria of the empirical distributions of plays of sequence prediction and change detection methods. The results show that they converge faster, and in more cases for change detection, than fictitious play.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Publications

This thesis is partly composed from three papers. In particular, chapters 4, 7, and 5 are based on the following conference and journal papers respectively.

Conference Proceedings

- Richard Mealing and Jonathan L. Shapiro [2013]. "Opponent Modelling by Sequence Prediction and Lookahead in Two-Player Games". In: *12th International Conference on Artificial Intelligence and Soft Computing*, pp. 385–396.
- Richard Mealing and Jonathan L. Shapiro [2015]. "Convergence of Strategies in Simple Co-Adapting Games". In: *Foundations of Genetic Algorithms XIII*.

Journal Papers

- Richard Mealing and Jonathan L. Shapiro [under review]. "Opponent Modelling by Expectation–Maximisation and Sequence Prediction in Simplified Poker". In: *IEEE Transactions on Computational Intelligence and AI in Games* (conditionally accepted).

Acknowledgements

I would like to thank everyone who has given me advice, encouragement, and support in my pursuit of this degree. This includes many people, from those who have helped me a great deal, to those who may have just had a conversation with me about this work. I have always found the interest of others very motivating. This work was made possible by the invaluable help of the following people: My supervisor, Jonathan L. Shapiro, whose guidance, support, and knowledge helped immensely in completing this work. My co-supervisor, Gavin Brown, whose help, particularly in the first year, allowed me to improve my ability to plan and communicate my research effectively. Joshua Knowles, for his interest in my work, especially his questions about the prisoner's dilemma and convergence in self-play. All of my colleagues, who were fantastic to work with, especially Hassan Bashir, David Fox, Darren Hau, Joe Mellor and Fanlin Meng for their questions, criticism, and ideas. My family and friends, whose encouragement and support helped see me through this project. Finally, I would also like to thank the Engineering and Physical Sciences Research Council [grant number EP/P505631/1] as well as the University of Manchester for their support.

Chapter 1

Introduction

Choosing how to act when faced with a decision is a problem everyone encounters every day. Ideally, we want to act in a way that will benefit us the most. This requires us to consider the consequences of not only our own actions, but also the actions of others. For example, if you are deciding how to get to work as fast as possible, then you may decide to drive along the shortest route, but if your co-workers make the same decision, then it may become congested and slower than alternatives. To complicate matters, the consequences of actions can manifest over different time periods. For example, perhaps driving along the shortest route is the fastest option, but it puts more strain on your vehicle causing it to break down faster. Additionally, we often face an incomplete view of the world, making us more uncertain about the state of the environment. For example, there might be unforeseen obstructions like road maintenance, or accidents, or spontaneous parades, etc. If we want to act optimally to achieve our goals, then we must anticipate how our interaction with the environment and others will proceed, taking into consideration the consequences of actions.

This chapter conveys the motivation behind this thesis, provides an outline of the problem, lists the contributions of this thesis, and describes its structure. The chapter begins in Section 1.1 by describing why we should study multi-agent learning.

1.1 Why Study Multi-Agent Learning?

Decision-making problems in the real-world often consist of multiple decision-makers. Studying multi-agent learning is therefore beneficial because it can be applied to these problems. Some examples include electronic trading (e.g. virtual market places like NASDAQ or Globex) and robotic interaction (e.g. search and rescue or exploration). We want agents that can learn to act rationally in the presence of other decision-makers. This means that we want them to learn to act optimally, where acting to benefit oneself given the information that you either know or could discover is optimal. The main problem is that the actions of each agent in a multi-agent system can affect the environment as well as the other agents. This causes an agent's optimal behaviour to often depend on how the other agents are behaving. For example, in a search and rescue operation, finding survivors is much more likely if each rescuer knows where the other rescuers are searching. Fortunately, this problem is extensively studied in game theory.

Game theory, traditionally used in economics, is well-suited to analyse interactions between rational decision-makers. There are two fundamental questions it can help us answer: How should an agent act, and how should an agent adapt, with other possibly influential decision-makers present? To help answer the first question, game theory provides us with a variety of solution concepts, which predict how a game will be played. Some solution concepts account for the consequences of agents' actions on the environment and other agents. Typically, an agent will have a value function that estimates the values of states, or actions, or both, and will use a strategy based on it. To help answer the second question, game theory research has found that even seemingly natural adaptation rules can lead to undesirable results, which is discussed in Section 3.2.3. This research is usually concerned with how to use the value function and any other relevant knowledge, such as opponent models, to change the agent's behaviour to maximise its expected rewards. Studying multi-agent systems, specifically games, can lead to improvements in the algorithms that operate within these domains, perhaps making them more applicable to real-world problems. This is becoming increasingly important given the shift in focus from single-agent learning to multi-agent learning due to the rise of distributed computing (e.g. the Internet).

This thesis considers simple multi-agent systems, which are games, usually with just two players. Each agent wants to maximise its own expected rewards. Thus, each agent can be considered greedy, not caring how well the other agent performs unless the other agent affects how well it performs. From the agent's perspective, its optimal behaviour is whatever will maximise its own expected rewards. A rational agent will try to achieve this optimal behaviour whilst factoring in its knowledge and potential knowledge. For example, if it knows that its opponent is perfect, then it would play the least exploitable strategy, or if it knows that learning more about its opponent would increase its expected rewards, then it would play more exploratively than exploitatively. Typically, there will be a finite amount of reward that can be divided between the agents, and each will act to receive the largest share. This will usually result in them acting as adversaries, but does not rule out cooperation if they can both benefit.

In Section 1.2, it is argued why it is beneficial to use games as the framework to develop agents, outlining the many advantages that they bring.

1.2 Why Study Games?

A game can be seen as a well-defined problem with specific rules and outcomes. Having a well-defined problem lets us focus on learning *how* to solve that problem rather than worrying about *what* the problem is. For this reason, games have been used as a testbed for artificial intelligence even before the advent of modern computing. For example, according to [*Chess Programming WIKI – Alan Turing* 2014], in 1952 Alan Turing developed a chess playing algorithm called Turochamp, and implemented it without a computer as a "paper machine" i.e. its computations were performed manually. Although in one of its only recorded matches it lost to one of Turing's colleagues, Alick Glennie, it was the beginning of a successful line of research. There are several aspects of games that make them useful for artificial intelligence research, including the following:

- Well-defined rules. Most games make it clear what players can and cannot do. For example, in chess, the bishop can only move diagonally. This simplifies implementing games and their players.

- Finite state and action spaces. Most games have a fixed number of states that they can be in, and a fixed number of actions each player can take in each state. For example, in checkers (draughts), the states are board configurations and the actions are legal moves (i.e. diagonal moves). Limited state and action spaces are conceptually simpler and easier to handle than large or infinite state and action spaces.

- Clear goals. The outcome of a game can always be categorised as either a win, or a draw, or a loss. Other measures of success may also exist such as how fast the game can be won, how much money a player can accumulate, how many pieces a player can take or preserve, etc. All of these goals are quantifiable. This makes it easy to evaluate how well a program works.

- Complex strategies. Finding the optimal strategy in a game can be challenging practically, or theoretically, or both. Thus, developing ways to find optimal strategies can push the boundaries of artificial intelligence research.
- Comparison agents. A measure of success lets us compare agents, and in turn, the effectiveness of different artificial intelligence techniques.
- Prior research. Games are a passion for many people and a popular research area. Consequently, there is a lot of work on analysing and finding solutions to many games, including many of the ones used in this thesis. This allows more time to be spent developing agents rather than analysing the games.
- Theoretical foundations. Game theory is a well-established field that can be used to help develop agents.

Developing algorithms to not only play games, but also to defeat humans in them, is a classic interest in artificial intelligence. Table 1.1 shows algorithms for games that have progressed around or beyond top human-level play.

Table 1.1: Artificial intelligence success in games.

Year	Game	Success
1979	Backgammon	BKG 9.8 beat world champion Luigi Villa [*Backgammon Programming*].
1994	Checkers	Chinook beat world champion Marion Tinsley [*Chinook vs. the Checkers Champ - Top 10 Man-vs.-Machine Moments - TIME*].
1995	Scrabble	Quackle beat former champion David Boys [*Scrabble Showdown: Quackle vs. David Boys - Top 10 Man-vs.-Machine Moments - TIME*].
1997	Chess	Deep Blue beat world champion Garry Kasparov [*IBM100 - Deep Blue*].
1997	Othello (Reversi)	Logistello beat world champion Takeshi Murakami [*Othello match of the year*].
2006	Go	Crazy Stone beat various pros [*CrazyStone at Sensei's Library*].
2008	Poker	Polaris beat various pros in heads-up limit Texas hold'em [*Man vs Machine II - Polaris vs Online Poker's Best*].
2011	Jeopardy!	Watson beat former winners Brad Rutter and Ken Jennings [*IBM computer Watson wins Jeopardy clash — Technology — theguardian.com*].
2015	Poker	Cepheus first to play an essentially perfect game of heads-up limit Texas hold'em [Bowling et al., 2015; *Cepheus Poker Project*].
2015	Various	Google DeepMind performs similar to professional human games tester at 49 arcade (Atari) games (e.g. Breakout, Space Invaders, etc) [Mnih et al., 2015]

Most of these examples are perfect information games, where each player knows all relevant information to make decisions except the other players' strategies. The exceptions are Scrabble, Poker, and Jeopardy!. However, although Scrabble has hidden information (the opponents' tiles), the champion program Quackle ignores this and treats the game as if it has perfect information. Thus, Table 1.1 represents a shift in focus from perfect information games to hidden information games. Hidden information makes learning how to act more difficult because the true state of the game may be hidden. This thesis looks at this more difficult case, with all games herein having hidden information.

Section 1.3 discusses, in general, the problem of learning to act, and describes the information requirements and goals of the approaches in this thesis.

1.3 Learning to Act

Ideally, we would like to maximise our total reward over the interaction time. This is challenging if we are not given any prior information about the consequences of our decisions, especially considering that the consequences may vary if other decision-makers alter their behaviours. In this situation, this leads to the question: How much time do we devote to learning the consequences? If we devote no time, and always act to maximise our reward given what little we know, then there is a high risk that our knowledge about the consequences will be wrong, and thus we will be acting incorrectly most of the time. On the other hand, if we devote all of our time, and always act to learn the consequences, then we will be maximising our reward very infrequently. Therefore, we need a balance between both.

An agent can use its rewards to directly guide its strategy by playing high-reward actions, or by playing to reach high-reward states. Many single-agent and multi-agent learning methods learn to act based solely on their rewards. Given that their rewards are usually determined by their opponents, then their learnt strategies often depend on their opponents' behaviours. In this way, they implicitly model their opponents in how much they value their actions or states. Due to this, it can be difficult to separate when an agent does well in its environment versus when it does well against its opponents. In comparison to an agent's rewards, it is less direct to use public actions to improve its strategy. The key is that the state is likely determined by all decision-makers' actions. By observing their actions, we can build models for how they will act. Knowing this allows us to anticipate future states and actions and to aim for those with the highest expected total rewards.

Section 1.4 explains in more detail what this thesis is about. It also lists the contributions of this thesis and gives its overall structure.

1.4 What is this thesis about?

This thesis is about modelling opponents whose actions can affect our rewards, and whose strategies may be based on memories of interaction, or may be changing, or both, and using these models to learn high-reward strategies in two-player (and one three-player) imperfect information games. These are games with hidden information. Opponents with memory-based or changing strategies are labelled as dynamic opponents, or opponents with dynamic strategies. To model these dynamic opponents' strategies, the focus is mainly on using sequence prediction methods. However, change detection methods are also looked at as an alternative to, or as a compliment to, sequence prediction methods. Chapters 4 and 5 develop approaches, which use opponent models learnt with sequence prediction methods, to learn high-reward strategies. The difference is that the games in Chapter 5 have more states and more hidden information than those in Chapter 4. In fact, in the games in Chapter 5, the opponent's hidden information may not even be revealed, but it is necessary in order to model the opponent, and so it is inferred using an online expectation-maximisation algorithm. This algorithm infers the opponent's hidden information based on its behaviour using an opponent model. The reason a sequence prediction method is used, rather than this opponent model on its own, is to model the dynamics of the opponent's strategy. Chapter 6 provides a preliminary investigation into using change detection methods to model these opponents by comparing the accuracies of variations of three state-of-the-art change detection methods at modelling changing categorical distributions. Finally, Chapter 7 investigates the convergence in self-play of the empirical distributions of plays of sequence prediction and change detection methods to mixed strategy Nash equilibria.

In Chapter 4, the initial focus of this thesis is on small simultaneous-move games, where the hidden information is the opponent's action, which is always revealed at the end of each game. Several sequence prediction methods are applied to model a variety of opponents, which include variable-order Markov models and finite automata, which have memory-based strategies, and state-of-the-art reinforcement learning algorithms, which have changing strategies. The idea in this chapter is to use these models, with (possibly discounted) rewards learnt with a standard reinforcement learning algorithm (Q-Learning), to lookahead with explicit tree search. The lookahead is meant to use the models and rewards to find action sequences with high expected rewards, which can then be followed.

This work is expanded in Chapter 5 to larger games, which have more states and more hidden information. Specifically, medium-size sequential-move games, where the hidden information is private information that the players' receive during the game (i.e. die-rolls, card deals), which may not be revealed at the end of each game. Given that the opponent's hidden information, which is necessary to model the opponent's strategy, is not always revealed, an online expectation-maximisation algorithm is developed to infer it. This algorithm infers the opponent's hidden information from its behaviour using an opponent model. However, instead of using this opponent model on its own, a sequence prediction method is also applied, with the inferred hidden information, to model the dynamics of the opponent's strategy. In experiments, a variety of opponents based on popular and state-of-the-art reinforcement learning and no-regret learning algorithms, which have changing strategies, are modelled. The idea in this chapter is to use the opponent model to iteratively improve the strategy of a state-of-the-art no-regret learning algorithm by simulating games between it and the model in order to allow it to gain higher rewards.

In Chapter 6, this thesis provides a preliminary investigation into using change detection methods to model these opponents as an alternative to, or as a compliment to, the sequence prediction methods. The idea in this chapter is to take variations of three state-of-the-art change detection methods, and to compare them against each other, and against an empirical distribution, at modelling changing categorical distributions. In the first comparison, the categorical distributions are independent and randomly change suddenly, or gradually, or a mixture of both. In the second comparison, the categorical distributions are the strategies of reinforcement learning agents and the models are used to play best-response strategies in matching pennies and rock-paper-scissors.

Finally, in Chapter 7, this thesis investigates if convergence in self-play of agents' empirical distributions of plays can be enhanced if each agent assumes that the other agents are changing their strategies over time. It looks at simultaneous-move, normal-form games with two or three actions, and two or three players. These games include generalised matching pennies, Shapley's game, and Jordan's game. Fictitious play, which assumes that the opponent uses a stationary strategy, is compared against two new variants that remove this assumption, and explicitly assume that the opponent uses a dynamic strategy. The opponent's strategy is predicted using a sequence prediction method in the first variant, and a change detection method in the second variant. Two hybrid methods are also proposed to improve the convergence of the sequence prediction and change detection methods. The first combines sequence prediction with fictitious play, whilst the second combines change detection with fictitious play.

1.4.1 Scope

The scope of this thesis is restricted to decision-making in simple multi-agent systems in the form of two-player games (as well as one three-player game). All the work within this thesis is about modelling opponents with memory-based or changing strategies, and using these models to learn high-reward strategies in the context of playing games. The overall approach adopted in this thesis is to conduct investigations that are largely experimental. Thus, it was necessary to design and develop algorithms to play within games to validate the ideas herein.

1.4.2 Contributions of this Thesis

The main contributions of this thesis are about modelling an opponent whose actions can affect our rewards, and whose strategy may be based on a memory of interaction, or may be changing, or both, and using this model to learn a high-reward strategy. Recall that an opponent with a memory-based or changing strategy is labelled as a dynamic opponent, or an opponent with a dynamic strategy (which is the reason for the thesis title). The contributions are as follows:

1. *An approach to learn high-reward strategies in small simultaneous-move games against opponents with memory-based or changing strategies.* This approach works by using sequence prediction to model the opponent and predict its actions, with reinforcement learning to learn the agent's own (possibly discounted) rewards, to explicitly lookahead via tree-search to find high-reward action sequences. An instance of the sequence prediction

method is created for each opponent information set within the game, and observes opponent actions across games. By doing so its predictions try to account for how the opponent's strategy is memory-based or changing. Empirical results show that this approach gains higher average rewards per game than state-of-the-art reinforcement learning agents when playing against variable-order Markov models in iterated rock-paper-scissors, finite automata in iterated prisoner's dilemma tournaments, and against each other in Littman's soccer game. These games are all small simultaneous-move games, where the hidden information is the opponent's action, which is always revealed at the end of each game. These results also demonstrate that several sequence prediction methods can effectively model opponents with memory-based or changing strategies. They support the idea of modelling these opponents with sequence prediction methods in general from areas such as data compression and string matching. (Chapter 4)

2. *An online expectation-maximisation algorithm that infers an agent's hidden information based on its behaviour in imperfect information games.* To do this, the algorithm uses a tuple of categorical distributions, one for each of the agent's information sets, to model its strategy. The expectation step infers the probability of the agent's actions that have been observed given its hidden information using the parameters of these categorical distributions, and then by Bayes' rule, the probability of its hidden information given its actions. The maximisation step updates the categorical distributions parameters by maximising their likelihood given the inferred distribution over the agent's hidden information. In general, as more of the agent's actions are observed, the accuracy of the inferred distribution over its hidden information will increase. Thus, this algorithm is used at the end of each game to try to maximise this accuracy. However, increasing the amount of hidden information in a game will make this distribution less accurate because there will be more instances of hidden information associated with the same behaviours, making them less distinguishable. (Chapter 5)

3. *An approach to learn high-reward strategies in medium-size sequential-move games against opponents with memory-based or changing strategies.* The approach works by using a model of the opponent learnt from sequence prediction, which requires the opponent's hidden information (inferred from

the online expectation-maximisation algorithm), to train a state-of-the-art no-regret learning algorithm by simulating games between it and the model. In these games the opponent's strategy usually depends on its hidden information, and so in these cases modelling its strategy requires its hidden information. Therefore, the online expectation-maximisation algorithm (contribution 2) is used to infer its hidden information based on its behaviour. An instance of the sequence prediction method is created for each opponent information set within the game, and observes opponent actions across games. By doing so its predictions try to account for how the opponent's strategy is memory-based or changing. The simulations against the anticipated opponent strategy allow the no-regret learning algorithm to learn regrets against it, which are essentially its anticipated future regrets. Empirical results show that this approach gains higher average rewards per game than the no-regret learning algorithm on its own when playing against popular and state-of-the-art algorithms in die-roll poker and Rhode Island hold'em. These games are medium-size sequential-move poker games, where the opponent's strategy often depends on hidden information, which is not always revealed at the end of each game. (Chapter 5)

4. *Demonstrating that several change detection methods can effectively model changing categorical distributions.* It is shown that amongst variations of three state-of-the-art change detection methods, some of them can effectively model changing categorical distributions. The results also show that they can be used to outperform state-of-the-art reinforcement learning agents in two simultaneous-move games. This supports the idea of modelling changing opponent strategies with change detection methods, either instead of, or with, sequence prediction methods. In the latter case, a change detection method could be used to model changes in a sequence prediction method's conditional distributions more effectively than it could on its own (if at all). Some change detection methods that are considered are modifications to be able to handle categorical distributions. Empirical results compare the accuracies of these methods against each other and against an empirical distribution at modelling changing categorical distributions. In the first comparison, the categorical distributions change either

suddenly, or gradually, or a mixture of both, and can be seen as being representative of changing opponent strategies. The results show that for few categories (2-5) the most accurate methods are BayesCPD-B and BayesCPD-C (sudden and mixed), ADWIN-D (gradual), and for many categories (10 or 20) the most accurate method is BayesCPD-C (sudden, gradual, and mixed). In the second comparison, the categorical distributions are the strategies of reinforcement learning agents, and the models are used to play best-response strategies against them. The results show that BayesCPD-B and BayesCPD-C generally produce the most accurate opponent models and the most rewarding best-response strategies. (Chapter 6)

5. *Experimental results for the self-play convergence to mixed strategy Nash equilibria of the empirical distributions of plays of sequence prediction and change detection methods.* It is shown that candidate sequence prediction and change detection methods converge faster than fictitious play. However, unlike fictitious play and change detection, sequence prediction does not always converge to the Nash equilibria in two-player, two-action, normal-form games derived from generalised matching pennies. Combining sequence prediction with fictitious play improves its convergence by reducing its convergence distance from the Nash equilibria for a number of these games. Combining change detection with fictitious play causes it to converge to the Nash equilibria in some of these games in fewer iterations, but overall decreases its convergence speed. Finally, it is also shown that, unlike fictitious play, the sequence prediction and change detection methods converge to the Nash equilibria in the difficult Shapley's and Jordan's games. (Chapter 7)

1.4.3 Thesis Structure

Chapter 2 formally defines the overall problem that this thesis tackles. The chapter begins in Section 2.1 by discussing one view of how an agent should act. This helps define the desirable behaviour for an agent to learn. Section 2.2 describes several beneficial multi-agent learning algorithm properties from the literature. Many of the algorithms that this thesis builds on, and compares against, incorporate these properties, and so understanding them helps in understanding how these algorithms work. Section 2.3 explains the assumptions that are made in the approaches in this thesis, as well as some assumptions that are not made, but that are often made in the literature. This helps to put this work in context and highlights the benefits and limitations of its approaches. Finally, Section 2.4 summarises this chapter. This chapter aims to provide the reader with a thorough understanding of the overall problem that the approaches in this thesis tackle as well as an understanding of where this work is placed within the literature, especially with regard to its assumptions.

Chapter 3 provides background and related work to help understand this work and its context. The chapter begins in Section 3.1 by exploring the relevant game theory. This includes formal definitions for games, their classes and categorisations, strategy categorisations, solution concepts (expanding on Chapter 2), and what it means to solve a game. It also describes the games used in this thesis as well as the bucketing abstraction used in Rhode Island hold'em. Section 3.2 discusses learning in games, describing repeated (iterated) games, which allows agents to learn, and stochasticity. It also looks at the problem of convergence that many multi-agent learning algorithms try to deal with whilst describing modes of convergence in general and in game theory. In addition, it describes machine learning in general and the relationship of this work to it, as well as providing formal descriptions of reinforcement learning, no-regret learning, and opponent modelling (focussing on sequence prediction) algorithms used within the approaches throughout. Lookahead and its importance is also explained. Finally, Section 3.3 summarises this chapter.

Chapter 4 addresses the first contribution of this thesis; can an opponent model learnt from a sequence prediction method be used, specifically its ability to make context based predictions, with rewards learnt from a reinforcement learning method, possibly with discounting, to lookahead using explicit tree search and find high-reward strategies? The chapter begins in Section 4.1 by describing sequence prediction and how it is used to model the opponent. This includes describing the core components of sequence prediction methods in general, which are their short-term and long-term memories, as well as giving brief descriptions of each sequence prediction method. Section 4.2 briefly describes how lookahead is involved in this approach, referencing back to Section 3.2 for a full explanation. Sections 4.3 and 4.4 outline the games and the opponents used in the experiments in this chapter respectively. At this point, in Section 4.5, the actual algorithm representing this approach is given. With the algorithm for this approach, and the opponents' algorithms in place, Section 4.6 empirically evaluates this approach. It reports and analyses the results from playing this approach in iterated rock-paper-scissors against variable-order Markov models, in iterated prisoner's dilemma tournaments against finite automata, and in a simplified soccer game against state-of-the-art reinforcement learning agents. In the first two games it compares the results of this approach against those obtained by state-of-the-art reinforcement learning agents, whereas in the third game there is a direct comparison. Finally, Section 4.7 summarises this chapter.

Chapter 5 addresses the second and third contributions of this thesis; can an opponent model learnt from a sequence prediction method, which itself uses an online expectation-maximisation algorithm to infer the opponent's hidden information that its strategy depends on, be used to train a state-of-the-art no-regret learning algorithm by simulating games between the algorithm and the model? This chapter builds on Chapter 4 by looking at larger games with more states and more hidden information. In Chapter 4 the hidden information is always revealed at the end of each game, whereas in this chapter the hidden information is not necessarily revealed at the end of each game. This is problematic as an opponent model requires this information, thus one of the contributions in this chapter is to derive an online expectation-maximisation algorithm that can infer the opponent's hidden information from its behaviour. The chapter begins in Section 5.1 by discussing opponent modelling in poker and relating it to the approach in this chapter. Section 5.2 discusses expectation-maximisation in general. Before empirically evaluating the approach in this chapter, the poker games and the opponents used in the experiments are defined in sections 5.3 and 5.4 respectively. A description of the approach is given in detail in Section 5.5. The stage is then set for Section 5.6, which gives the empirical results of comparing this approach against the comparison agents. Finally, Section 5.7 summarises this chapter.

Chapter 6 addresses the fourth contribution of this thesis; can change detection methods be used to model changing categorical distributions? The chapter begins in Section 6.1 by discussing the problem of learning a changing categorical distribution. It moves on in Section 6.2 to describe several variations of three state-of-the-art change detection methods, some of which are first proposed in this thesis, that can be used to model changing categorical distributions. In general, they do this by trying to maintain a window of observations that are only drawn from the current distribution, and then using this window of samples to form an empirical distribution. Once these various change detection methods have been defined, an empirical comparison is performed to compare their accuracies in Section 6.3. This involves testing their accuracies at modelling changing categorical distributions. Section 6.4 discusses how change detection methods could be used with sequence prediction methods, and how this could potentially improve them, which in turn could improve the approaches in chapters 4 and 5. Finally, Section 6.5 summarises this chapter.

Chapter 7 addresses the fifth and final contribution of this thesis; can convergence in self-play be enhanced if each agent assumes that the other agents are changing their strategies over time. To help answer this, fictitious play, which assumes that the opponent uses a stationary strategy, is compared against two new variants proposed in this chapter that remove this assumption and explicitly assume that the opponent uses a dynamic strategy. The opponent's strategy is predicted using a sequence prediction method in the first variant, and a change detection method in the second variant. The chapter begins in Section 7.1 with a motivational example outlining how fictitious play acts in self-play. Section 7.2 describes extensions to fictitious play to help put this work into context. This leads to Section 7.3, which discuss how sequence prediction and change detection methods could improve on this. This section also explains two hybrid methods, the first combining sequence prediction with fictitious play, and the second combining change detection and fictitious play. Section 7.4 shows experimental results. Finally, Section 7.5 summarises this chapter.

Chapter 8 presents the overall conclusions for what has been learnt about improving multi-agent learning through the approaches of this thesis in Section 8.1, and suggests possible directions for future research in Section 8.2. Most of the future work suggestions are to do with removing the assumptions that have been made throughout this work, whilst the rest are about reconciling this work with other promising approaches in the literature.

Chapter 2

Problem Definition

This thesis looks at the simplest type of multi-agent system, which is one with only two agents. Each of these agents is acting in its own self-interest to maximise its own rewards. However, one difficulty is that each agent's rewards can be affected by the other agent's actions. This makes it difficult for an agent to anticipate what rewards it can expect to receive in the future for whatever sequence of actions it chooses because this often requires it to know what will result from its own as well as its opponent's actions.

This chapter defines the problem in more detail. Section 2.1 begins by looking at how an agent should act. Included in this is a description of a spectrum of strategies, with a maximally exploitative best-response strategy at one end, and a risk-averse Nash equilibrium strategy at the other end.

2.1 How Should an Agent Act?

There are many views on how an agent should act and for the most part the answer depends on the task at hand. Ideally, we want an agent to achieve its goals, but there may be many ways to do this. One popular set-up is to give an agent a reward, which is a numerical feedback that is positive either for achieving a goal or for getting closer to achieving a goal, zero for having no effects on the goal, and negative otherwise. Another popular set-up is to give an agent an error (or regret), which is a numerical feedback that is positive and closer to zero the closer an agent is to achieving a goal. This leads us to two common views of how an agent should act, which are firstly, that an agent should act to maximise its rewards, and secondly that an agent should act to minimise its errors. A rational agent is defined as one that acts optimally in its own self-interest given its knowledge and potential knowledge. A rational agent would choose its actions according to its beliefs such that they maximise its expected reward, where an expected reward is a reward multiplied by the probability of receiving that reward. If an agent always does this, then it is playing an estimate of a best-response strategy. The accuracy of the best-response strategy would depend on the accuracy of the agent's beliefs. This implies that a rational agent would learn to play an estimate of a best-response strategy.

Unfortunately, a best-response strategy is usually unknown and difficult to calculate because it depends on other agents' strategies, which are usually unknown, as well as random events in the environment. A best-response strategy could be estimated, but Johanson, Zinkevich, and Bowling, 2008 showed that a bad estimation could lead to a much lower expected reward than that of the best-response strategy. This is because there is no guarantee that a strategy close to a best-response strategy in the strategy space will yield similar expected rewards. For example, imagine playing rock-paper-scissors repeatedly against an opponent. In the first game you observe the opponent playing rock. If you assume that the opponent's strategy can be determined from its past play, then you might predict that its strategy is to always play rock. Therefore, your best-response strategy is to always play paper. However, in the second game when you play paper, your opponent plays scissors and you lose. In this way, a best-response strategy that looks highly profitable can become disastrous if the assumptions behind it are wrong. The incorrect assumption in this example is that the opponent's strategy can be determined from a single past game. Another problem is that it is usually difficult to calculate a best-response strategy in large games because you have to consider every decision point. A best-response strategy can be seen as an offensive strategy as it tries to get the most for the agent. In a two-player zero-sum game, it would simultaneously maximise the agent's expected reward, and minimise the opponent's expected reward, completely exploiting the opponent.

Alternatively an agent could play a Nash equilibrium strategy (i.e. its strategy from a Nash equilibrium strategy profile), which is a sort of safe-strategy against rational opponents, which are worst-case opponents in zero-sum games. A Nash equilibrium is a tuple of strategies, one per agent, where no single agent can change its strategy to increase its expected reward. In other words, each agent is playing a best-response strategy to the other agents' strategies. One advantage of an agent playing a Nash equilibrium strategy in a two-player game is that it minimises its risk by guaranteeing at least its expected reward at the Nash equilibrium, which is usually much higher than the minimum reward in the game. Another advantage of it for a two-player game is that it is stable because whatever strategy the opponent uses, this guarantee is still there. If the opponent is not playing its half of the Nash equilibrium, then the agent can only receive either an equivalent or greater expected reward. A disadvantage is that, if the opponent is not playing its half of the Nash equilibrium, then there will usually be a strategy (such as a best-response strategy) that will give a higher expected reward. Also, if there are two or more opponents, then a group of them might be able to change their strategies simultaneously and lower the agent's expected reward. Another difficulty is choosing a Nash equilibrium in a general-sum game where many may exist. Finally, a Nash equilibrium is also difficult to compute, although it can be computed offline. A Nash equilibrium strategy can be seen as a defensive strategy against rational opponents, which in the case of zero-sum games are opponents that play worst-case strategies. In a two-player zero-sum game, a Nash equilibrium strategy is equivalent to a minimax strategy.

The approaches in this work attempt to learn best-response strategies to try to maximise total expected reward. To account for the typical dependence of the best-response strategy on the opponent's strategy, the approaches in chapters 4 and 5 model the opponent's strategy by observing its behaviour using sequence prediction methods. In chapters 6 and 7 change detection methods are also looked at for modelling opponent strategies. It is unlikely that these methods will be able to learn the opponent's strategy perfectly, especially if the opponent is changing its strategy. Therefore, the approaches in chapters 4 and 5 do not completely rely on their opponent models. The approach in chapter 4 does this by only using its opponent model to lookahead up to a limited depth, and the approach in chapter 5 does this by only using its opponent model to simulate possible interactions, not to predict exactly how the opponent will act. However, chapters 6 and 7 do use best-response strategies because, Chapter 6 is specifically looking at the accuracy of change detection methods, and Chapter 7 is looking at convergence in self-play analogous to fictitious play, which uses best-response strategies. This work mainly deals with relatively small games where opponent models can approximate the opponent's strategy within a reasonable time. However, for larger games, it can take a long time. Chapter 5 has to deal with this problem, and does so by abstracting the game along with the opponent's strategy space, down to a smaller more manageable size.

Section 2.2 reviews the various properties that have been suggested in the literature as desirable for multi-agent learning algorithms. The approaches in this thesis do not explicitly focus on these properties. However, many of the comparison agents are designed around incorporating these properties, and by understanding them, it aids in understanding how they work.

2.2 Learning Objectives

An agent attempting to learn in a multi-agent environment faces several challenges. The main problem is the co-adaptive nature of the system. If an agent changes its strategy, then other agents may react and change their strategies. This violates the assumption, commonly used in single-agent learning problems, that a fixed optimal strategy exists. Instead, as agents change their strategies, each agent's optimal strategy usually changes. Due to this, many learning algorithms with stable dynamics in single-agent learning environments become unstable in multi-agent environments. Another problem is that explicitly calculating optimal or best-response strategies requires knowledge of other agents' strategies, which are usually private. Finally, out of the learning algorithms that consider other agents, many assume that their strategies are stationary, in that their action choices only depend on the environment's state. However, agents may change their strategies suddenly, or gradually, or periodically, or in a combination of ways. Their strategies may also be conditioned on some information such as a history of states, or actions. If the other agents are learning, then it is very likely that they will change their strategies and so ideally this should be taken into account.

Bowling and Veloso, 2001, 2002 attempted to formalise the idea that a learning agent should learn a best-response strategy when possible by proposing rationality and convergence as desirable properties of multi-agent learning algorithms.

Rationality: An agent is rational if it learns a best-response strategy when the other agents learn stationary strategies.

A rational agent was defined in Section 2.1 as one that acts optimally in its own self-interest given its knowledge and potential knowledge, such that it plays actions with the maximum expected reward according to its beliefs. To satisfy that definition, this rationality property would be a necessary but not sufficient condition. This is because if all other agents' strategies are fixed, then playing a best-response strategy against them would maximise an agent's expected rewards. However, if the other agents' strategies are not fixed, then to satisfy the definition from Section 2.1 an agent may have to learn different best-response strategies.

Convergence: A set of rational agents playing against each other should converge to a stationary strategy profile.

If all agents are rational according to the definition from Section 2.1, then each one should learn to use a best-response strategy. If all agents simultaneously use best-response strategies, then by definition they will be at a Nash equilibrium. Therefore, as all agents learn their best-response strategies, they should converge to a Nash equilibrium, at which point no single agent can change its strategy to increase its expected rewards, making it a stationary strategy profile. A necessary but not sufficient condition for an algorithm to have this property is that it will converge against itself. This criterion can also be seen as a form of co-evolution or co-adaptation.

Additions to this pair of desirable properties for multi-agent learning algorithms have been proposed by Powers and Shoham, 2005 as well as by Butterworth, 2010. Powers and Shoham, 2005 proposed that for any choice of $\epsilon > 0$ and $\delta > 0$ there should exist a number of rounds (iterations of the game) T_0, which is polynomial in $1/\epsilon$, $1/\delta$, the number of outcomes k, and the maximum possible difference in rewards across the outcomes b, such that for a number of rounds, $t > T_0$, the algorithm achieves an average reward of at least:

1. $V_{\text{BR}} - \epsilon$ against any member of a target set of opponents with probability $1 - \delta$, where V_{BR} is the expected reward of a best-response strategy against that opponent.

2. $V_{\text{selfPlay}} - \epsilon$ in self-play with probability $1 - \delta$, where V_{selfPlay} is the minimum reward achieved by any Nash equilibrium that is not Pareto dominated by another Nash equilibrium.

3. $V_{\text{security}} - \epsilon$ against any opponent with probability $1 - \delta$, where V_{security} is the agent's security (or maximin) reward for the stage game.

The first property says that an agent should asymptotically learn a best-response strategy against any opponent belonging to a target set. The second property says that an agent should asymptotically converge to a Nash equilibrium that is not Pareto dominated in self-play. Finally, the last property says that an agent should asymptotically converge to a strategy that at least gives the security value of the stage game (i.e. the game being repeated) against any opponent. Butterworth, 2010 proposed that:

4. An agent should maximise the amount of time spent playing a best-response strategy in order to maximise the expected rewards it receives.

A weaker condition than the last is that only when playing against a slow-learning (or quasi-stationary) opponent should the agent track the opponent's strategy and attempt to spend as much time as possible playing a best-response strategy against it. Either condition is only feasible if the agent has enough computational resources to track its opponent, and to learn a best-response strategy. It is important to note that for any learning agent it may be possible to create another learning agent that can learn sufficiently quickly to defeat it. For example, continuing improvements in hardware and software will allow agents to learn strategies in the same amount of time but using finer abstractions for games with large state spaces such as poker, which will probably lead to improved play.

Section 2.3 lists the overall assumptions that are made throughout and within the approaches in this thesis. These assumptions are about the types of games played, as well as the information available in agent interactions.

2.3 Assumptions

In order to explain this work, the assumptions about the games used, as well as the way agents interact, must be explained. For the most part, it is assumed that agents play in finite, competitive, two-player, general-sum games with imperfect information. In actuality, most of the games are zero-sum, and there is also one three-player game, which is highlighted when it is used. Generally, the games the agents in this thesis play have the following features:

- Two-players, with the only exception being Jordan's game (three-players).
- At each game state, each player receives a real-valued reward, although these rewards are usually zero except at terminal game states.

- General-sum, the sum of players' rewards at the end of the game can be any value, although in most games the sum is zero (zero-sum).
- There is at least one point in the game where at least one of the players has information hidden from it, preventing it from differentiating between multiple decision points.
- There is no explicit communication between players, they cannot directly pass information to each other (the games do not allow this). The only information a player can see about another player is its public actions.
- A player's rewards depend on prior actions, which can include player decisions, opponent decisions, and stochastic events (e.g. card deals or die-rolls).

With regards to how agents interact, the assumptions are as follows:

- It is assumed that players know the rules of the game, meaning that they know the following:
    1. The actions that they can take at each of their decision points.
    2. The rewards that they will receive in each game state.

This first part of this assumption is necessary because a player cannot play the game, or describe its strategy, unless it knows what its available actions are. The second part of this assumption is also necessary because a player cannot learn unless it has some measure of success. Note that knowing your reward for each game state does not guarantee that when playing the game you will be able to reach a particular game state or even be aware of what the game state is.

- It is assumed that the game is played repeatedly.

This assumption is also necessary for learning. If a game is only played once, then a player is unlikely to learn much, especially if the game has stochastic events. Technically, during a single game, a player could alter its strategy for later decisions based on the results in earlier decisions, but this requires intermediate rewards and only makes sense if these decision points are related in some way.

- It is assumed that each player has the option to have perfect recall.

A player with perfect recall never forgets revealed information. Although a player may not require perfect recall, it is not restricted from having it.

The prior three assumptions are the only ones placed on interaction. To help differentiate this work within the literature, it is important to point out some assumptions that are not placed on interaction, which are as follows:

- It is not assumed that players know other players' strategies.

Knowledge of other players' strategies is commonly provided in the literature on imperfect information games. The rationalisation for this is that such knowledge could be easily estimated through repeated interaction. However, Butterworth, 2010 showed that this assumption is not entirely valid, and that learning an estimate can have consequences.

- It is not assumed that each player treats each game independently, or that each player's strategy cannot be based on a memory of interaction, or that each player's strategy cannot change.

It is often assumed in the literature that a player's strategy consists of a stationary distribution for each state where it acts, such that for a particular state where it acts, it chooses an action by randomly sampling from its associated distribution. However, a player may act based on experience i.e. a memory of interaction, such that these distributions may be conditional on past information. For example, in the iterated prisoner's dilemma, tit-for-tat is a strategy that initially cooperates and then copies the opponent's previous action, thus for the single state in the prisoner's dilemma where tit-for-tat acts, its strategy consists of five conditional probabilities $\Pr(C|\cdot) = 1$, $\Pr(C|C) = 1$, $\Pr(C|D) = 0$, $\Pr(D|C) = 0$, $\Pr(D|D) = 1$, where C is cooperate and D is defect. One way to handle this, which is used by this thesis, is to model the conditional distributions. Another way is to include the conditional information as part of the state space, which allows unconditional distributions to be associated with each state. An agent may also change its strategy, especially if it is learning. By not assuming that each player's strategy is memoryless and stationary, this thesis is dealing with a wider range of, and more realistic players. For example, humans and animals base their decisions on memories of interaction, and rational players often change their strategies.

2.4 Chapter Summary

This chapter has explained the overall problem that the approaches in this thesis face with respect to learning in an environment with multiple agents. It has defined a spectrum of behaviours from best-response strategies to Nash equilibrium strategies that rational agents might consider trying to learn. The approaches in this thesis in chapters 4 and 5 are designed to learn towards best-response strategies, whilst not completely relying on them to account for possible inaccuracies. Whilst the players in chapters 6 and 7 intentionally use best-response strategies to measure accuracy and to converge in self-play respectively. Additionally, this chapter has provided an overview of a variety of desirable multi-agent learning properties suggested in the literature, which many of the agents that this thesis builds on and compares against try to incorporate. Finally, this chapter has discussed the overall assumptions that this work makes, and does not make, about the types of games played, as well as the accessible information during interactions. In short, this thesis studies learning high-reward strategies against other agents whose strategies may be memory-based, or changing, or both, by modelling them to help anticipate, and have better control over, future rewards.

Chapter 3

Background and Related Work

This chapter gives background material and related work to help understand and put the work in this thesis into context. In particular, it explains how the components in the approaches of this thesis work and what they are based on in the literature. The chapter begins in Section 3.1 by exploring the relevant game theory. This includes formally defining a game and its categorisations as well as defining different strategy categorisations. Additionally, it takes a closer look at solution concepts that were touched on in Chapter 2. Finally, it describes the games used throughout and the bucketing abstraction used in Rhode Island hold'em. Section 3.2 discusses the environments that agents in this thesis learn in, which are repeated games. It also explains several definitions of convergence and their relation to game theory as well as the problem of convergence, which is a problem that many multi-agent learning algorithms tackle. Finally, it describes and explains the reinforcement learning, no-regret learning, and opponent modelling (with a focus on sequence prediction) algorithms that the approaches in this thesis use, in addition to the importance and the use of lookahead.

3.1 Game Theory

Game theory is a branch of applied mathematics that studies strategic decision-making. Specifically, it is the study of mathematical models of interaction (i.e. conflict and cooperation). These interactions are usually studied between decision-making agents who are both intelligent and rational. Game theory has applications in many fields such as economics, politics, war, psychology, logic, and biology. Modern game theory examines the decisions of human, animal, and artificial players where the focus is often on the search for or the convergence to solution concepts. The framework of game theory is extremely useful for multi-agent systems where an agent's optimal strategy usually must account for the strategies of other agents, which can influence that agent's rewards.

3.1.1 What is a Game?

A game can be defined as a structured interaction of one or more players where each player is trying to accomplish goals according to a set of rules. The outcome of a game generally depends on the decisions of the player(s) as well as any stochastic events described by the rules of the game (e.g. die rolls or card deals). Each player is usually given indications of its performance through reward signals, which are typically received at the end of a game (e.g. $+1$ for a win, 0 for a draw, and -1 for a loss), but could be received at any point during the game. The reinforcement learning, dynamic programming, and game theory communities sometimes use different terminology to describe the same concepts about games. Table 3.1 shows the synonyms between the terminologies in these communities. Throughout this work some of these terminologies are used interchangeable, mainly the synonyms between the reinforcement learning and game theory communities.

Table 3.1: Comparison and definition of synonyms between terminologies in the reinforcement learning, dynamic programming, and game theory communities.

Reinforcement Learning	Dynamic Programming	Game Theory	Definition
Agent	Controller	Player	The entity (e.g. person, computer, animal, etc.) interacting with the system.
Environment	Process	Game	The system in which the entities interact.
State	State	State	The information that represents the system.
Action	Control	Move	Something an entity does to influence the system.
Policy	Policy	Strategy	A mapping from states to the choices available in those states.
Deterministic policy	Deterministic policy	Pure strategy	A mapping from states to single choices that are always taken in those states.
Stochastic policy	Stochastic policy	Mixed strategy	A mapping from states to probability distributions over the choices available in those states.
Immediate reward	Immediate cost	Immediate payoff	Numerical value received upon transitioning to another state.
Utility	Cost	Payoff	Numerical value received upon reaching a terminal state.
Return	Cost-to-go	Return	Numerical value based on some aggregate of the transition values.

3.1.2 Game Classes

Cooperative Classes

The word "game" typically implies a competitive game (e.g. to game a person is to exploit that person). In competitive (or non-cooperative) games, the goals of at least one player are opposed to the goals of at least one other player. The players are usually prohibited from communicating with each other in order to prevent them from cooperating by forming coalitions. There has been a lot of research into cooperative and coordination games. Much of this research has focused on learning to form coalitions and allocating tasks between coalition members. Interesting applications include the self-organisation of robots in search and rescue missions [Scerri et al., 2010] and sampling extraterrestrial rocks [Chien et al., 2000]. This work focusses on competitive games and does not consider the possibility of players explicitly communicating or sharing rewards. It is still possible for a player to implicitly communicate through its behaviour, but if this leads to any form of cooperation it will likely be because the player believes it is in its best interest.

Payoff Classes

Players' payoffs provide one way to classify games. In a zero-sum game, players' payoffs must sum to zero. In other words, one player's gain must equal a loss to the other players. It follows that if one player's payoff is p, then the other players must share (perhaps unequally) a payoff of $-p$. This makes zero-sum games strictly competitive because each player's goals are diametrically opposed to the other players' goals. All outcomes in a zero-sum or constant-sum game are Pareto optimal since no player can increase its payoff without at least one other player lowering its payoff. Games where all outcomes are Pareto optimal are also referred to as conflict games. In constant-sum games, players' payoffs always sum to the same value and so zero-sum games are a type of constant-sum game. Any constant-sum game can be converted into a zero-sum game by shifting all its payoffs by an amount equal to the negative of the constant divided by the number of players. Explicit examples of zero-sum games include rock-paper-scissors and poker. Any n-player game where only one player can win and where payoffs are not specified can be implicitly considered as a zero-sum game because you can assume that the player who wins gets a payoff of 1 whilst the players who lose each get a payoff of $-1/n$. Implicit examples of zero-sum games include chess and checkers. Even a game with multiple winners can be zero-sum if the losers share the negative of the sum of the winners' payoffs.

In a general-sum game, players' payoffs at the end can sum to any amount. This means that for any two outcomes of the game, the sums of the players' payoffs at each outcome can be different values. It is even possible that each player's payoff is completely independent of the other players' payoffs. General-sum games encompass all possible payoff structures and can be seen as a superclass of constant-sum games, which in turn can be seen as a superclass of zero-sum games.

Information Classes

A game can be classified according to the information it makes available to its players. There are two main information classes: perfect information games, and imperfect information games. The difference between them is that in the latter, one or more players have game-related information hidden from them. This is important because the amount of information a player has can greatly affect the complexity of the game from its perspective and the correctness of its decisions. If either there is too much information in a game such that the player has to omit some of it, or if a game already has information hidden from the player, and the omitted or hidden information affects the agent's rewards either directly or indirectly, then this can introduce more uncertainty about the correctness of its decisions. If a player omits or ignores information in a game, then this reduces the complexity of the game from its perspective and is called an abstraction. Technically an abstracted game is a different game from the original game. An example could be to abstract a poker game that uses a standard fifty-two card deck to ignore card suits. This simple abstraction would drastically reduce the size of the game and the amount of information, but would make the player unable to determine if it had a flush. The result would be that any rewards it receives from having a flush would appear to be random. This is a major problem with using abstractions, they usually make the player less able to correctly assign credit (or blame) when a reward dependent on the hidden information is received.

Perfect Information A perfect information game is one in which every player can observe the exact state of the game at any point in the game. The exact state of a game describes the game structure and all previous moves made by all players. If the game is represented in extensive-form (as a tree), then this means that every player can see the exact node the game is at in the tree at any point. Chess is a classic example of a perfect information game because it has no hidden information, each player can observe all moves as well as the arrangement of the pieces on the board. In other words the players can, if they choose to remember, know everything there is to know about the state of the game at all points. Typically the only information a player is unaware of is the other players' strategies, but this is not part of the game itself. When playing a perfect information game, the optimal set of actions is always deterministic. This means that for whatever game state a player is in, there is set of equal actions that are definitely better than every other action the player can take at that point. Note that perfect information games can still involve stochasticity. For example, backgammon is a perfect information game because, like chess, players can observe all moves and can see the arrangement of the pieces on the board. However, it has stochasticity due to the uncertainty of die rolls.

Imperfect Information An imperfect information game is one in which there is at least one point where at least one of the players is unsure of the game state. For a game represented in extensive-form (as a tree), if it has imperfect information, then there would be at least one point where at least one of the players does not know the exact node the game is in. This occurs because some information is hidden from one of the players. For example, any normal-form game is an imperfect information game because the players take their moves simultaneously, meaning that at the moment a player takes its move, the other players' moves are hidden from it. Another example of an imperfect information game would be a poker-like game where, for at least most of the game, the other players' hands are hidden. An imperfect information game has the property that at least one information set contains at least two game states (information sets are defined in Section 3.1.3). Different amounts of imperfect information provide varying levels of complexity for learning algorithms. The approaches in this thesis view all games using extensive-form representation, which requires the normal-form games to be viewed as sequential games with imperfect information. This is done by arbitrarily selecting the move order and hiding each player's move until all players have moved.

3.1.3 Game Categorisations

Many algorithms used as comparisons and to develop the approaches within this thesis are designed to learn to act optimally in a Markov Decision Process (MDP). Thus, firstly a MDP is defined to help understand these algorithms. Secondly, normal-form games are defined, which are used throughout, particularly in chapters 4 and 7. Thirdly, stochastic games are defined, which generalise MDPs and normal-form games. One stochastic game, namely Littman's soccer game, is used in Chapter 4. Fourthly, extensive-form games are defined, which are used throughout, particularly in Chapter 5. Fifthly, repeated games are defined since all games herein are repeated to allow agents to learn. Finally, sequence-form games are defined because, although they are not used, they can provide a more efficient strategic description than normal-form and extensive-form games and may be useful for future research. To illustrate some of the different types of game categorisations the same game is shown in normal-form, extensive-form, and sequence-form in Figure 3.1.

Markov Decision Process (MDP)

A MDP is a tuple (S, A, P, R) where:

- S is a finite set of states.
- A is a finite set of actions.
- $P : S \times A \times S \rightarrow [0, 1]$ is a function that gives the probability that an action will lead from one state to another.
- $R : S \times A \times S \rightarrow \mathbb{R}$ is a function that gives the immediate reward received for playing an action that leads from one state to another.

A MDP can be seen as a one-player, multi-state game. At each state in the game the player chooses an action and chance probabilistically determines the next state based on the current state and chosen action. The difference between a MDP and a one-player extensive-form game is the following. A MDP could be defined such that any state leads to any other state. Whereas in an extensive-form game, the states and the transitions between them must be arranged in a tree such that each state only has one parent that leads to it, except for the root or start state, which has no parent. A MDP tuple can also be defined to include a discount factor $\gamma \in [0, 1]$, which determines how long-term reward is calculated from immediate rewards. However here if a discount factor is used, then it is assumed to be part of the agent's specification.

Normal-Form Game

An n-player normal-form (matrix) game is a tuple $(N, \mathcal{A}, \mathcal{R})$ where:

- $N = \{1, 2, \ldots, n\}$ is a finite set of players.
- $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ is a finite set of finite sets of actions (or pure strategies), where A_i is player i's finite set of actions. For example, in rock-paper-scissors, $A_1 = A_2 = \{R, P, S\}$.
- $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ is a finite set of payoff functions (payoff matrices), where R_i is player i's payoff function, which maps from the joint action space to the space of real numbers $R_i : A = (A_1 \times A_2 \times \cdots \times A_n) \rightarrow \mathbb{R}$.

To play, each player, i, selects an action, $a_i \in A_i$, and receives a payoff equal to the entry in its payoff matrix at the position of the joint action, $a = (a_1 \in A_1, a_2 \in A_2, \ldots, a_n \in A_n) \in A$. A normal-form game can be seen as a multi-player, single-state game. Each dimension of a normal-form game payoff matrix corresponds to a player's set of actions. This representation is easy to understand for small games like rock-paper-scissors, or the prisoner's dilemma, but becomes less intuitive for larger games. Although any extensive-form game can be represented in normal-form, determining the set of pure strategies for each player can be time consuming. This is because, in general, the size of a normal-form game payoff matrix can be exponential in the number of terminal nodes in the corresponding extensive-form game tree. An advantage of normal-form representation is that it is easy to handle mathematically, as long as the payoff matrices are not too large. Note that "reduced" normal-form is the same as normal-form except without duplicate strategies.

Stochastic Game

An n-player stochastic game is a tuple $(N, S, \mathcal{A}, P, \mathcal{R})$ where:

- $N = \{1, 2, \ldots, n\}$ is a finite set of players.
- S is a finite set of states.

- $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ is a finite set of finite sets of actions or pure strategies such that A_i is player i's finite set of actions.
- $P : S \times A \times S \rightarrow [0, 1]$ where $A = (A_1 \times A_2 \times \cdots \times A_n)$ is a function that gives the probability that a joint action will lead from one state to another.
- $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ is a finite set of payoff functions such that $R_i : S \times A \times S \rightarrow \mathbb{R}$ is player i's payoff function that gives its immediate reward for a joint action that leads from one state to another.

The concept of a stochastic game was introduced by Lloyd Shapley in the early 1950s. A stochastic game can be seen as a multi-player, multi-state game. At each state in the game all players act simultaneously and chance probabilistically determines the next state based on the current state and chosen joint action. This is very similar to a MDP, except in a MDP there is only one player.

Extensive-Form Game

An extensive-form game involves sequential decision-making. It can be visualised as a game tree, with nodes as game states and edges as actions. At each non-terminal node a player is "on turn", which means that it chooses the action to take at that node. The chosen action determines the edge that is followed to the next node. Each node, h, has only one parent and so can be represented by a unique history or sequence of actions taken to reach it, $h = (a_1, a_2, \ldots, a_m)$, where each action, a_i, $1 \leq i \leq m$, is taken by one of the players. These actions include "chance" actions such as die rolls or card deals, which are taken by the "chance" player. Thus, h represents all the information seen by an omniscient observer. An n-player extensive-form game is a tuple $(N, H, Z, P, A, \mathcal{I}, \sigma, u)$ where:

- $N = \{1, 2, \ldots, n\} \cup \{c\}$ is a finite set of players including a chance player.
- H is a finite set of all possible nodes or histories, which are action sequences that contain the empty sequence and every prefix of a sequence. For example, in rock-paper-scissors, $H = \{(), (R), (P), (S), (R, R), (R, P), (R, S), (P, R), (P, P), (P, S), (S, R), (S, P), (S, S)\}$, or in die-roll poker an element could be, $h = (⚁, ⚃, \text{raise}) \in H$.
- $Z \subseteq H$ is a finite set of terminal histories or leaf nodes. For example, in rock-paper-scissors this set is $Z = \{(R, R), (R, P), (R, S), (P, R), (P, P),$

$(P, S), (S, R), (S, P), (S, S)\}$, or in die-roll poker an element could be, $h = (\text{⚁}, \text{⚃}, \text{raise}, \text{fold}) \in Z$.

- $P : H \rightarrow N$ is a one-to-one function mapping from histories to players such that $P(h) = i \in N$ is the player who takes an action after history $h \in H$.

- $A : H \rightarrow \mathcal{A}$ is a one-to-one function mapping from histories to sets of actions such that $A(h) = \{a_1, a_2, \ldots, a_{|A(h)|}\} \in \mathcal{A}$ is the set of actions available to player $P(h)$ after history $h \in H$, and $(h, a_i) \in H$ for all $1 \leq i \leq |A(h)|$. For example, in rock-paper-scissors, $A((R)) = A((P)) = A((S)) = \{R, P, S\}$, or in die-roll poker, $A((\text{⚁}, \text{⚃}, \text{raise})) = \{\text{raise}, \text{call}, \text{fold}\}$.

- $\mathcal{I} = \{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_n\}$ is a finite set of information partitions, one per player. Player i's information partition is a finite set of its information sets, $\mathcal{I}_i = \{I_1, I_2, \ldots, I_{|\mathcal{I}_i|}\}$. Player i's information set, $(I \subseteq H) \in \mathcal{I}_i$, is a finite set of histories that are indistinguishable from player i's perspective. The term information partition is used because each node belongs to exactly one information set and there are no empty information sets. If there is no hidden information, then each node belongs to its own information set. Note that, in the games considered, for any information set I, $P(I) = P(h)$ and $A(I) = A(h)$ for any $h \in I$. For example, in rock-paper-scissors, $\mathcal{I}_1 = \mathcal{I}_2 = \{\{(R, R), (R, P), (R, S)\}, \{(P, R), (P, P), (P, S)\}, \{(S, R), (S, P), (S, S)\}\}$, or in die-roll poker one element in player 1's information partition could be $I \in \mathcal{I}_1 = \{(\text{⚀}, \text{⚀}, \text{fold}), (\text{⚀}, \text{⚁}, \text{fold}), (\text{⚀}, \text{⚂}, \text{fold}), (\text{⚀}, \text{⚃}, \text{fold}), (\text{⚀}, \text{⚄}, \text{fold}), (\text{⚀}, \text{⚅}, \text{fold})\} \in \mathcal{I}_1$.

- $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ is a strategy profile, which is a finite tuple of strategies, one per player. Player i's strategy is a one-to-one function mapping from its information sets where it acts to discrete probability distributions over the actions available at those information sets, $\sigma_i : \{I : I \in \mathcal{I}_i \text{ and } P(I) = i\} \rightarrow \Delta(A(I))$, where $\Delta(\cdot)$ is the space of probability distributions over a set. Player i's strategy can also be seen as a set of discrete probability distributions, $\sigma_i = \{f_{A(I)} : I \in \mathcal{I}_i \text{ and } P(I) = i\}$, where $f_{A(I)}$ is a probability mass function over $A(I)$.

- $u = \{u_1, u_2, \ldots, u_n\}$ is a finite set of utility functions, one per player. Player i's utility function is a one-to-one function mapping from histories (usually terminal) to its rewards (real numbers), $u_i : H \rightarrow \mathbb{R}$.

To play at any non-terminal history, $h \in (H \setminus Z)$, the player on turn, $P(h)$, samples and plays one of its available actions, $a \in A(h)$, according to its strategy, $\sigma_{P(h)}(I)$, where $h \in I$ and $I \in \mathcal{I}_{P(h)}$. Once sampled, this action a is appended to the history, h, forming a new history, $(h, a) \in H$. The game is played from the root, $h = ()$, to a terminal history, $h = z \in Z$. At each history, $h \in H$, each player, i, receives a payoff according to its utility function of $u_i(h)$.

An extensive-form game can be seen as a multi-player, multi-state game. It is arguably the most intuitive representation because it is easy to visualise the progression within a game as a descending path through its tree. However, it is less flexible than a stochastic game because it cannot easily represent loops in a game. In fact, if there are loops in a game such that previous states can be freely revisited, then the corresponding game tree would be infinite. Nodes where the chance player is on turn generally have a fixed probability distribution over the available actions. Although an extensive-form game tree represents decisions (nodes) sequentially, information sets can be used to represent simultaneous moves. For example, if a player acts at the same time as its opponent, then the player would not know which path its opponent followed in the game tree. Therefore, the player's information set would contain nodes (histories) representing all possible paths the opponent could have taken.

Repeated or Iterated Game

A repeated game (also known as a supergame or an iterated game) is a game built by repeating some base game (also called a stage game). If a player's rewards depend on other players' actions in the stage game, then the consequence of playing a repeated game with that stage game is that the player must take into account how its actions will affect the other players' future actions. Repeated games can be broadly classified as either infinitely or finitely repeated. A player may act very differently depending on when it thinks the game will end. Specifically, if a player has the opportunity to exploit an opponent, but could face retaliation if it does so, then it might be tempted to wait until near the end of the game when the opponent will no longer have the opportunity to retaliate. Note that a Nash equilibrium in a stage game may not be a Nash equilibrium in a repeated game using that stage game.

Sequence-Form Game

Sequence-form is a more recently invented notion than normal-form or extensive-form, and was introduced by Koller, Megiddo, and Stengel, 1994 as an efficient way to construct linear programs and linear complementarity problems that can solve extensive-form games with perfect recall. A player with perfect recall never forgets revealed information. The inspiration for sequence-form originated from the observation that although extensive-form is a more succinct, and arguably more natural representation than normal-form, many techniques to solve games are more applicable in normal-form. The root of the problem is that the number of pure strategies in a game that is represented in extensive-form is usually exponential in the size of its game tree [Stengel, 1996]. For example, the number of pure strategies in a game tree where every inner node (game state) has the same number of actions (branching factor), b, is b^d, where d is the depth of the tree. Thus, converting a game represented in extensive-form into a game represented in normal-form in order to solve it is computationally intractable for anything but small games. Sequence-form tackles this problem by constructing a strategy space linear in the size of the game tree. This allows solution techniques for normal-form games to be performed directly on sequence-form games, which circumvents the computationally intensive conversion into normal-form.

In sequence-form, a player takes each node of the game tree and considers the choices needed to reach it. These sequences of choices take the place of pure strategies the player would use in normal-form. However, a player cannot just choose a single sequence like a pure strategy because it does not define how to act in any situation. Instead, to define a strategy, the player assigns realisation probabilities to each sequence. A binary assignment of realisation probabilities then defines a pure strategy. The main benefit of this method is that a player's expected payoff is linear in the realisation probabilities for its sequences. This allows the construction of linear equations representing an optimisation problem that is directly solvable. There are already efficient solution algorithms to solve these optimisation problems and therefore sequence-form allows exponentially faster solutions to be obtained for games represented in extensive-form. This is because the conversion to sequence-form is faster than the usually exponential time necessary to convert to normal-form.

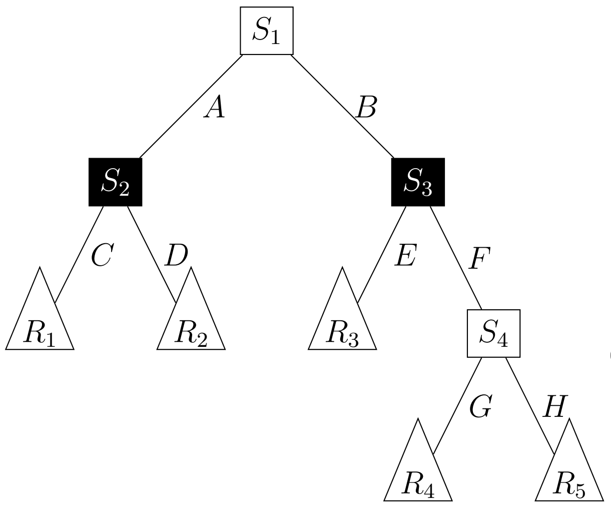
A sequence-form representation of an n-player extensive-form game with perfect recall is a tuple (N, S, x, u) where:

- $N = \{1, 2, \ldots, n\}$ is a finite set of players.

- $S = \{S_1, S_2, \ldots, S_n\}$ is a finite set of finite sets of sequences of actions, one per player. Player i's set of sequences of actions, S_i, contains each sequence of actions that it would take to reach each of its information sets in the game tree (disregarding the actions of the other players). For example, in rock-paper-scissors, $S_1 = S_2 = \{\emptyset, R, P, S\}$.

- $x = \{x_1, x_2, \ldots, x_n\}$ is a finite set of realisation plans. Player i's realisation plan, $x_i : S_i \to [0, 1]$, is a function mapping from its sequences of actions to realisation probabilities (or realisation weights). For any player, $i \in N$, $x(\emptyset) = 1$, and $x(s_I) = \sum_{a \in A(I)} x(s_I a)$, where $s_I \in S_i$ is player i's sequence of actions taken to reach its information set $I \in \mathcal{I}_i$ and $s_I a \in S_i$ such that it plays the actions in s_I and then plays action a. Player i's probability for taking action a is $\frac{x_i(s_I a)}{x_i(s_I)}$. Player i's realisation probability for a sequence of actions, $x_i(s_I)$, is the product of the probability of each of its actions in that sequence, i.e. if $s_I = (a_1, a_2, \ldots, a_m)$, where some actions may be hidden from player i, then $x_i(s_I) = \prod_{j=1}^{m} \Pr(a_j | (a_1, a_2, \ldots, a_{j-1}))^{b_j}$, where $b_j = 1$ if a_j is one of player i's actions, otherwise $b_j = 0$. Note that here an information set is as defined the same as for an extensive-form game.

- $u = \{u_1, u_2, \ldots, u_n\}$ is a finite set of payoff functions (sparse payoff matrices), one per player. Player i's payoff function, u_i, is a one-to-one function mapping from its sequences of actions to rewards (real numbers), $u_i : \sigma_i \to \mathbb{R}$.



(a) Extensive-form representation.

(b) Normal-form representation.

	CE	DE	CF	DF
AG	R_1	R_2	R_1	R_2
AH	R_1	R_2	R_1	R_2
BG	R_3	R_3	R_4	R_4
BH	R_3	R_3	R_5	R_5

(c) Sequence-form representation.

	∅	C	D	E	F
∅	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
A	(0,0)	R_1	R_2	(0,0)	(0,0)
B	(0,0)	(0,0)	(0,0)	R_3	(0,0)
BG	(0,0)	(0,0)	(0,0)	R_3	R_4
BH	(0,0)	(0,0)	(0,0)	R_3	R_5

Figure 3.1: An arbitrary game in extensive-form, normal-form, and sequence-form. In extensive-form representation, each node is a game state with white squares as player one's decision points, black squares as player two's decision points, white triangles as terminal game states, and edges as actions. In normal-form representation, each row is a pure strategy for player one, each column is a pure strategy for player two, and each entry for a specific row and column are the players' rewards. Finally, in sequence-form representation, each row is a sequence of actions for player one, each column is a sequence of actions for player two, and each entry for a specific row and column are the players' rewards. An empty action sequence is denoted by $\emptyset$. Also, most entries have zero rewards (sparse encoding) because the combination of each player's action sequence at these points either leads to a non-terminal node, or is unrealisable. Note that R_i is a pair of rewards, $R_i = (p_i, q_i)$ where p_i is player one's reward and q_i is player two's reward.

3.1.4 Strategy Categorisations

In this work agents' strategies are represented as behaviour strategies. However, it is easier to understand how behaviour strategies work when compared to other representations. Thus, the three main types of strategy representations are described here, which are pure strategies, mixed strategies, and behaviour strategies. An agent's strategy describes how that agent will act. A strategy is usually specified for a particular domain such as a game. In the case of a game, a player's strategy specifies how that player chooses an action at any state in the game where that player acts. If a player decision point is encountered and that player's strategy is not specified for that decision point, then the player will not know what to do and the game will halt.

Pure Strategy

A pure strategy is a deterministic strategy and so it has no probabilistic choices. An agent with a pure strategy will always play the same action in the same state. An example of a pure strategy for player one with regard to Figure 3.1 is AG, where player one will always play action A in state S_1 and will always play action G in state S_4. If player one played this pure strategy, then it would never actually reach state S_4. However, a pure strategy requires a player to specify what action it would play in each state where it acts even if some of those states cannot be reached.

Mixed Strategy

A mixed strategy is a probability distribution over pure strategies. A totally mixed strategy is a mixed strategy where each pure strategy has a strictly positive probability of being selected. An agent with a mixed strategy will sample and execute a pure strategy according to the probability distribution its mixed strategy assigns over pure strategies. An example of a (totally) mixed strategy for player one with regard to Figure 3.1 is to play each pure strategy with equal probability i.e. $\Pr(AG) = \Pr(AH) = \Pr(BG) = \Pr(BH) = 0.25$. An agent using a mixed strategy only makes one randomised choice before the game starts, which is to select its pure strategy according to its mixed strategy probabilities.

Behaviour Strategy

A behaviour strategy specifies a probability distribution over the actions available to an agent at each of its decision points. An agent with a behaviour strategy will sample and execute an action at each of its decision points that are visited according to its probability distribution associated with that decision point. An example of a behaviour strategy for player one with regard to Figure 3.1 is to play action A with probability 0.8, action B with probability 0.2, action G with probability 0.4, and action H with probability 0.6. It would be difficult to use a behaviour strategy for a game represented in normal-form because this representation only shows the players' pure strategies (each row/column is a pure strategy) and does not show decision points. Instead, a behaviour strategy is more suited to extensive-form representation. An agent using a behaviour strategy makes a randomised choice at each of its decision points in a game in order to select an action. A behaviour strategy is usually more efficient than a mixed-strategy. For example, in Figure 3.1, a mixed-strategy for player one requires three probabilities ($\Pr(AG)$, $\Pr(AH)$, and $\Pr(BG)$ since $\Pr(BH) = 1 - \Pr(AG) + \Pr(AH) + \Pr(BG)$), whereas a behaviour strategy for player one requires two probabilities ($\Pr(A)$ and $\Pr(G)$ since $\Pr(B) = 1 - \Pr(A)$ and $\Pr(H) = 1 - \Pr(G)$).

Kuhn's Theorem - Relating Mixed and Behaviour Strategies

Kuhn's theorem establishes a relationship between mixed strategies and behaviour strategies. The theorem states that if a player has perfect recall, then for any one of its behaviour strategies there is an equivalent mixed strategy in the sense that their expected payoffs are the same [Kuhn, 1953]. A player with perfect recall never forgets revealed information. By discovering this relationship, Kuhn showed that a behaviour strategy can be represented as a mixed strategy and vice versa. Consequently, with perfect recall, the set of Nash equilibria is the same when using mixed or behaviour strategies.

3.1.5 Solution Concepts

The ideal is to develop agents that will perform as well as possible according to some performance metric in every situation they are placed in. Ultimately, the performance metric that this thesis is most concerned with is the agent's cumulative reward over all the games that it plays. This can also be looked at as its average reward per game. This thesis looks at developing agents that use predictions from models of opponents with memory-based or changing strategies to find, or at least approach, best-response strategies. For the following definitions, there is a finite set of players, $N = \{1, 2, \ldots, n\}$. Each player, $i \in N$, has a finite set of pure strategies, A_i, and a utility function that maps tuples of pure strategies, where each tuple contains one pure strategy per player, to rewards (real numbers), $u_i : \prod_{j=1}^{n} A_j \to \mathbb{R}$. Each player i also has a strategy, $\sigma_i \in \Sigma_i \equiv \Delta(A_i)$, where $\Delta(\cdot)$ is the space of probability distributions over a set. The strategy profile, σ, is defined as the tuple containing each player's strategy, $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n) \in \Sigma = \prod_{j=1}^{n} \Sigma_j$, and σ_{-i} as the same as σ but excluding player i's strategy, $\sigma_{-i} = (\sigma_1, \sigma_2, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n) \in \Sigma_{-i} = \prod_{j=1, j \neq i}^{n} \Sigma_j$. Finally, player i's expected reward for the strategy profile σ is defined as $\overline{u}_i(\sigma) \equiv \sum_{a \in \prod_{j=1}^{n} A_j} u_i(a) \prod_{i=1}^{n} \sigma_i(a(i))$, where $a(i)$ is player i's pure strategy in a.

Best-Response

A best-response strategy for a player is a strategy that results in the most preferred outcome for that player against the other players' current strategies. Formally, a best-response strategy for player i, $\sigma_i^* \in \Sigma_i$, is a strategy which, given the strategies of all other players $\sigma_{-i} \in \Sigma_{-i}$, results in player i's most preferred outcome such that

$$\sigma_i^* \in \arg\max_{\sigma_i \in \Sigma_i} \overline{u}_i(\sigma_i, \sigma_{-i}) \tag{3.1}$$

Here the most preferred outcome for a player is the outcome that maximises its expected reward. If a mixed strategy is a best-response strategy, then each pure strategy in the mix with positive probability must also be a best-response strategy. The reasoning for this is as follows: If it were not true, then there would be at least one pure strategy with positive probability that has a lower expected reward than the mixed strategy. Furthermore, if the probability of this pure strategy was decreased to zero and the probabilities of the other pure strategies that have positive probabilities were increased, then this must raise the expected reward of the mixed strategy. However, this means that the original mixed strategy would not have been a best-response strategy because it would have a lower expected reward than the new mixed strategy. This is a contradiction. The main problem with using a best-response strategy is that if the environment changes, or if the opponents' strategies change, then it may become a very bad strategy as shown by Johanson, Zinkevich, and Bowling, 2008.

Maxmin and Minmax

A maxmin strategy is one that maximises a player's worst-case expected reward. Formally, a maxmin strategy for player i is defined as

$$\arg\max_{\sigma_i}\min_{\sigma_{-i}}\overline{u}_i(\sigma_i,\sigma_{-i}). \tag{3.2}$$

The worst-case expected reward to player i for the maxmin strategy is also known as the maxmin value (security level) of the game and is defined as

$$\max_{\sigma_i}\min_{\sigma_{-i}}\overline{u}_i(\sigma_i,\sigma_{-i}) \tag{3.3}$$

Player i's maxmin strategy is the strategy that it would commit to in order to maximise its expected reward if it was forced to announce its strategy first and then all the other players chose their strategies to minimise player i's expected reward.

A minmax strategy is one that minimises an opponent's best-case expected reward. Formally, a minmax strategy for player i is defined as

$$\arg\min_{\sigma_i}\max_{\sigma_{-i}}\overline{u}_{-i}(\sigma_i,\sigma_{-i}). \tag{3.4}$$

The best-case expected reward to the opponents $-i$ for the minmax strategy is also known as the minmax value of the game and is defined as

$$\min_{\sigma_i}\max_{\sigma_{-i}}\overline{u}_{-i}(\sigma_i,\sigma_{-i}). \tag{3.5}$$

The minimax theorem states that for every two-player zero-sum game with a finite number of strategies for each player, there exists a value V (called the value of the game) and a (possibly mixed) strategy for each player such that player one's expected reward is at least V, independent of player two, and player two's expected reward is at most V, independent of player one. If $V = 0$, then the game is said to be fair, otherwise it is said to be unfair in that $V > 0$ favours player one and $V < 0$ favours player two. The minimax solution is a Nash equilibrium for these games. A strategy profile that satisfies the equation

$\min_{\sigma_i} \max_{\sigma_{-i}} \overline{u}_{-i}(\sigma_i, \sigma_{-i}) = \max_{\sigma_i} \min_{\sigma_{-i}} \overline{u}_i(\sigma_i, \sigma_{-i})$ is called a minimax solution. In other words, the minmax solution is equal to the maxmin solution.

Nash Equilibrium

A Nash equilibrium is a tuple of strategies, one per player, where each player's strategy, σ_i^*, is a best-response strategy to the other players' strategies, σ_{-i}^*, [Nash, 1950] such that

$$\overline{u}_i(\sigma_i^*, \sigma_{-i}^*) \geq \overline{u}_i(\sigma_i, \sigma_{-i}^*) \text{ for all } \sigma_i \in \Sigma_i \text{ and for all } i \in N \tag{3.6}$$

At a Nash equilibrium, no player can increase its expected reward by changing its strategy unilaterally. However, it may be possible for two or more players to change their strategies simultaneously to increase their expected rewards. Nash, 1950 proved that if players can use mixed strategies, then at least one Nash equilibrium exists for all finite n-player games, where finite means that there is a finite number of pure strategies for each player.

In a zero-sum game, a player's expected reward at each Nash equilibrium, or mixture of Nash equilibria, is the same. This is called the value of the game, and is the same as the value from the minimax theorem. This is not the case in general-sum games where different Nash equilibria, or mixtures of different Nash equilibria, can give a player different expected rewards. Therefore, players in a general-sum game can suffer from the problem of having to choose between different Nash equilibria. It is also possible that a mixture of Nash equilibria in a general-sum game is not a Nash equilibrium strategy.

An ϵ-Nash equilibrium is an approximate Nash equilibrium. It is a tuple of strategies, one per player, where each player's strategy, σ_i^*, is within ϵ of a best-response to the other players' strategies, σ_{-i}^*, in terms of expected rewards, i.e.

$$\overline{u}_i(\sigma_i^*, \sigma_{-i}^*) + \epsilon \geq \overline{u}_i(\sigma_i, \sigma_{-i}^*) \text{ for all } \sigma_i \in \Sigma_i \text{ and for all } i \in N \tag{3.7}$$

where $\epsilon \geq 0$ is some non-negative real-valued number. If $\epsilon = 0$, then only a Nash equilibrium satisfies this equation. If $\epsilon = \infty$, then any strategy profile satisfies this equation. Thus, although an ϵ-Nash equilibrium is called an approximate Nash equilibrium, it is only approximate if ϵ is small with respect to the range of the expected rewards.

Regret Minimisation

Regret (also called opportunity loss or difference regret) is generally defined as the difference between the rewards that a player could have received by playing some strategy and the rewards that the player did receive by playing its actual strategy. There are many concepts of regret. For a game represented in normal-form, Zinkevich, 2004, 2005 defines external regret, swap regret, internal regret, external response regret, and internal response regret.

Take a normal-form game, which is iterated T times, and let h be the history of actions played in these games i.e. $h = ((a_1^1, \ldots, a_n^1), \ldots (a_1^T, \ldots, a_n^T))$ where a_i^t is player i's action at time t. Furthermore, let R_i be player i's payoff function such that given the actions of all players it returns player i's payoff, A_i be the set of available actions to player i and a_{-i}^t be the actions of all players except player i. Player i's external regret, r_{ext}, is the difference between its total payoff and the maximum total payoff it could have got by replacing all its actions with a single fixed action whilst assuming that the other players' actions are the same

$$r_{\text{ext}} = \max_{a_i^* \in A_i} \sum_{t=1}^{T} \left[R_i(a_i^*, a_{-i}^t) - R_i(a_i^t, a_{-i}^t) \right]. \tag{3.8}$$

Player i's swap regret, r_{swa}, is the difference between its total payoff and the maximum total payoff it could have got by replacing each of its actions using a function $f : A_i \to A_i$ whilst assuming the other players' actions are the same

$$r_{\text{swa}} = \max_{f} \sum_{t=1}^{T} \left[R_i(f(a_i^t), a_{-i}^t) - R_i(a_i^t, a_{-i}^t) \right]. \tag{3.9}$$

Player i's internal regret, r_{int}, is the difference between its total payoff and the maximum total payoff it could have got by replacing all plays of a particular action a_i' with another action a_i^* whilst assuming the other players' actions are the same

$$r_{\text{int}} = \max_{a_i', a_i^* \in A_i} \sum_{t: a_i^t = a_i'} \left[R_i(a_i^*, a_{-i}^t) - R_i(a_i^t, a_{-i}^t) \right]. \tag{3.10}$$

A problem with relying on these regrets to learn high-reward strategies is that they do not consider the consequences of an agent's actions. In particular, they ignore how other agents may react. For example, always defecting in the prisoner's dilemma will have zero external, swap, and internal regrets against any agent, suggesting it is a good strategy. However, always defecting, rather than always cooperating, will result in a much lower total payoff against strategies such as grim trigger and tit-for-tat. Zinkevich, 2004, 2005 proposed external and internal response regrets to account for the immediate and short-term consequences of an agent's actions.

For a game represented in extensive-form, Zinkevich et al., 2008 define overall regret and counterfactual regret. Take an extensive-form game, which is iterated T times. Let σ_i^t be player i's strategy at time t, σ_{-i}^t be the tuple of opponents' strategies at time t. Player i's overall regret, r_{ove}, is defined as the difference between the total expected reward of its actual strategy at each time step, and the maximum total expected reward it could have got with a single fixed strategy

$$r_{\text{ove}} = \max_{\sigma_i^* \in \Sigma_i} \sum_{t=1}^{T} \left[\overline{u}_i(\sigma_i^*, \sigma_{-i}^t) - \overline{u}_i(\sigma_i^t, \sigma_{-i}^t) \right] . \tag{3.11}$$

Furthermore, let $\pi_{-i}^{\sigma^t}(I)$ be the probability of reaching information set I according to the strategy profile at time t, but ignoring player i's strategy, $\overline{u}_i(\sigma^t|I)$ be the same as $\overline{u}_i(\sigma^t)$ except it is calculated from information set I, and $\sigma^t|_{I\to a}$ be the same as σ^t except player i always plays action a at information set I. Player i's counterfactual regret, r_{cou}, at an information set, I, is defined as the difference between the total expected reward of its actual strategy at each time step at that information set, and the maximum total expected reward it could have got by playing a fixed action for all time steps at that information set, weighted by the probability of reaching that information set if it had tried to do so

$$r_{\text{cou}} = \max_{a \in A(I)} \sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I) \left[\overline{u}_i(\sigma^t|_{I\to a}, I) - \overline{u}_i(\sigma^t, I) \right] . \tag{3.12}$$

Counterfactual regret is explained in more detail in Section 3.2.9. According to Zinkevich et al., 2008, the following theory connects the concept of overall regret and a Nash equilibrium strategy: In a two-player zero-sum game at time T, if each player's average overall regret is less than ϵ, then their average strategies are a 2ϵ Nash equilibrium. Thus, as each player's average overall regret approaches zero, their average strategies approach a Nash equilibrium.

Iterated Elimination of Dominated Strategies

Given two strategies σ and σ' belonging to a player, σ dominates σ' if the player gets higher payoffs with σ than with σ' regardless of what the other players do. A dominated strategy can be removed because a rational player would not play it. By removing a dominated strategy, other strategies may be revealed that are also dominated. If all dominated strategies are removed, and only one strategy remains for each player, then they are a Nash equilibrium.

Pareto Optimality

Pareto optimality or efficiency is an allocation of resources amongst individuals such that any reallocation cannot make an individual better off without making another individual worse off. Thus, a strategy profile is Pareto optimal if no player can increase its expected payoff without another player decreasing its expected payoff. In any zero-sum or constant-sum game all strategy profiles are Pareto optimal. Generally, if there are a finite number of outcomes, then at least one Pareto optimal solution will exist. A Pareto optimal strategy profile can be seen as a solution concept because there is no wasted payoff (i.e. no payoff that could have been gained by one or more players at no cost to the other players). A strategy profile that contains best-response, or maxmin, or minmax, or regret minimising strategies, or that is a Nash equilibrium, is not necessarily Pareto optimal. A strategy profile can also be described as Pareto optimal within a subset of strategy profiles e.g. a Nash equilibrium can be Pareto optimal compared to other Nash equilibria. Finally, an outcome is Pareto dominated if it is not Pareto optimal and if a Pareto optimal outcome exists.

3.1.6 Solving Games

According to Allis, 1994, stating that a game is solved usually indicates that a property with regard to the outcome of that game has been determined. For two-player zero-sum games with perfect information he provides three definitions for the degree to which a game has been solved: ultra-weakly solved, weakly solved, and strongly solved.

Ultra-Weakly Solved An ultra-weakly solved game is one where, for any initial positions, the game-theoretic value of that state has been determined. This implies that, for any initial positions, it is known what the outcome (i.e. win, draw, or loss) will be for all players if they were to play perfectly.

Weakly Solved A weakly solved game is one where, for any initial positions, a strategy has been determined that will obtain at least the game-theoretic value of that state, for each player, under reasonable resources. This implies that, for any initial positions, it is known what the outcome will be for all players if they were to play perfectly and there is an algorithm that can achieve this outcome.

Strongly Solved A strongly solved game is one where, for any legal positions, a strategy has been determined that will obtain the game-theoretic value of that state, for each player, under reasonable resources. This implies that, for any legal positions, it is known what the outcome will be for all players if they were to play perfectly and there is an algorithm that can achieve this outcome.

The definitions of a weakly solved game and of a strongly solved game mention that their strategies should obtain the game-theoretic value, for each player, under reasonable resources. The idea is that a game is only considered weakly solved or strongly solved if the corresponding algorithm can be run by existing, perhaps state-of-the-art, hardware in a reasonable time. If this were not the case, then it could be argued that, for example, chess is weakly solved using the minimax algorithm. Realistically it is known that, on existing hardware, this would be far too computationally expensive in terms of time and memory. There is an ordering amongst these three definitions. If a game is strongly solved, then it is also weakly solved, and if a game is weakly solved, then it is also ultra-weakly solved.

In a two-player zero-sum game, a Nash equilibrium corresponds to playing the safest strategy because it minimises the opponent's maximum payoff. Since the game is zero-sum, where whatever the opponent gains the player loses and vice versa, this also corresponds to maximising the player's minimum payoff. This assumes that the opponent always acts to minimise the player's payoff and thus maximise its own. In the above definitions, perfect play corresponds to playing a Nash equilibrium strategy, which is the same as a minimax strategy. The game-theoretic value is the expected payoff of a Nash equilibrium strategy, which will be the same for all Nash equilibria. This means that finding a Nash equilibrium strategy profile would be sufficient to strongly solve one of these games.

In order to find a Nash equilibrium in a two-player zero-sum game the minimax principle can be used alongside linear programming. By expressing the game in normal-form or sequence-form equations can be formed corresponding to a linear optimisation problem. Linear programming applied to these equations can then optimise for the maximum value of the player's minimum expected payoff. The constraints would be that the resulting strategy describes valid probability distributions over actions. The solution would describe mixed strategies for both players that form a Nash equilibrium. There is also a method to find a Nash equilibrium in a two-player general-sum game represented in normal-form (as a bi-matrix). This method requires the problem to be formulated as a linear complementarity problem, which is a special case of quadratic programming.

Analytically finding a Nash equilibrium is generally considered to be a hard problem. Although linear programming can be used to find a Nash equilibrium in polynomial time for a two-player zero-sum game, there are no efficient polynomial time algorithms for finding a Nash equilibrium in a non-zero sum game that the author is aware of. Due to this, one might be tempted to assume that this problem is non-polynomial (NP)-complete. However, there is a peculiarity that, unlike other NP-complete problems, Nash's theorem guarantees that there is always at least one (possibly mixed strategy) Nash equilibrium in a game with a finite number of actions. In fact, Daskalakis, Goldberg, and Papadimitriou, 2006 as well as Chen, Deng, and Teng, 2009 proved that finding a Nash equilibrium is Polynomial Parity Arguments on Directed graphs (PPAD)-complete, where PPAD is a subclass of NP. The advantage of this is that a PPAD-complete problem is less likely to be intractable than an NP-complete problem. On the other hand, Conitzer and Sandholm, 2008 showed that determining if a Nash equilibrium has a specific property out of a certain large set of properties is NP-hard. These properties include, for example, having a specific social welfare (sum of expected payoffs), or as another example, determining if a specific pure strategy occurs in the support (has positive probability). Not only this but they also showed that maximising certain properties, such as social welfare, is inapproximable. Given the computational complexity of analytically finding Nash equilibria, machine learning is a tempting alternative for discovering approximate solutions.

3.1.7 Games in the Experiments

This section describes the games used throughout this thesis. Many of these games are represented in normal-form but a stochastic game and two poker games represented in extensive-form are also used. All of these games are two-player except one, which is Jordan's game. A large part of the focus is on zero-sum games, but some general-sum games are also used. In all the games it is assumed that the players have perfect recall. These games are used to test the performance of the approaches in this thesis with the exceptions of Shapley's game and Jordan's game, which are used to test the self-play convergence to mixed strategy Nash equilibria of players' empirical distributions of plays.

Rock-Paper-Scissors

Rock-paper-scissors or roshambo is a two-player, three-action, zero-sum, normal-form game. Each player can play: rock which counters (smashes) scissors, paper which counters (smothers) rock and scissors which counters (cuts) paper. There is a unique mixed strategy Nash equilibrium, which is for each player to play each of its actions with equal probability of 1/3. Due to its popularity, there have been a variety rock-paper-scissors tournaments for both humans [*2009 World RPS Championships — World RPS Society*] and algorithms [*Rock Paper Scissors Programming Competition*]. The payoff matrix is shown in Table 3.2.

Table 3.2: Rock-paper-scissors payoff matrix, each entry shows (row player payoff, column player payoff) for the given row and column actions.

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

Shapley's Game

Shapley's game is a two-player, three-action, general-sum, normal-form game. It is rock-paper-scissors but modified such that negative payoffs are replaced with zero payoffs (see Table 3.2), which turns it into a general-sum game. It retains the unique mixed strategy Nash equilibrium of rock-paper-scissors, which is for each player to play each of its actions with equal probability of 1/3. Finally, it has been used as an example of where fictitious play fails to converge in self-play [Shapley, 1963] and to test the self-play convergence of other algorithms to a unique mixed strategy Nash equilibrium [Abdallah and Lesser, 2008; Zhang and Lesser, 2010]. The payoff matrix is shown in Table 3.3.

Table 3.3: Shapley's game payoff matrix, each entry shows (row player payoff, column player payoff) for the given row and column actions.

	R	P	S
R	0,0	0,1	1,0
P	1,0	0,0	0,1
S	0,1	1,0	0,0

Miscoordination game

Miscoordination game is a two-player, two-action, general-sum, normal-form game. Each player can select one of two actions. If they both select different actions, then they each get a payoff of one, otherwise they each get a payoff of zero. There are two pure strategy Nash equilibria, which are both players playing different actions. There is also a range of mixed strategy Nash equilibria, in each one a player's probability of playing an action is equal to the other player's probability of playing the other action. Fudenberg and Kreps, 1993 used a miscoordination game to show that although fictitious play appears to converge to a Nash equilibrium in it, the expected Nash equilibrium payoff may not obtained. The payoff matrix is shown in Table 3.4.

Table 3.4: Miscoordination game payoff matrix, each entry shows (row player payoff, column player payoff) for the given row and column actions.

	A	B
A	0,0	1,1
B	1,1	0,0

Matching Pennies

Matching pennies is a two-player, two-action, zero-sum, normal-form game. Each player can select either heads or tails. The goal of the first (row) player is to match the coin face of the second (column) player, whereas the goal of the second (column) player is the opposite, to mismatch the coin face of the first player. There is a unique mixed strategy Nash equilibrium, which is for each player to play each of its actions with equal probability of 1/2. This is a well-known game that can be seen as a simpler version of rock-paper-scissors with one less action per player. The payoff matrix is shown in Table 3.5.

Table 3.5: Matching pennies payoff matrix, each entry shows (row player payoff, column player payoff) for the given row and column actions.

	H	T
H	1,-1	-1,1
T	-1,1	1,-1

Jordan's Game

Jordan's game [Jordan, 1993] is a three-player, two-action, general-sum, normal-form game. It is an extension of matching pennies to include a third player. Each player can select either heads or tails. The first player wants to match the coin face of the second player, the second player wants to match the coin face of the third player, and the third player wants to mismatch the coin face of the first player. There is a unique mixed strategy Nash equilibrium, which is for each player to play each of its actions with equal probability of 1/2. The game has been used to test the self-play convergence to mixed strategy Nash equilibria of various algorithms [Abdallah and Lesser, 2008; Bowling and Veloso, 2002; Zhang and Lesser, 2010]. Its rewards are shown in Table 3.6.

Table 3.6: Jordan's game (also known as three-player matching pennies) payoff matrix, each entry shows (player one payoff, player two payoff, player three payoff). Player one selects the outer row, player two selects the column, and player three selects the inner row.

		H	T
H	H	1,1,-1	-1,-1,-1
	T	1,-1,1	-1,1,1
T	H	-1,1,1	1,-1,1
	T	-1,-1,-1	1,1,-1

Prisoner's Dilemma

The prisoner's dilemma is a two-player, two-action, general-sum, normal-form game where both players are caught after committing a crime together. Each player is moved to a separate interrogation room and has two actions. A player can either defect against the other player, which means confessing to its crime, or can cooperate with the other player, which means remaining silent. If both players defect, then there is enough evidence to give them both the maximum sentence, but these are reduced slightly for their admissions. If one player defects and the other cooperates, then the player who defected is set free since the case hinged on that testimony whereas the player who cooperated receives the maximum sentence. Finally, if both players cooperate, then there is only enough evidence to give them both short sentences. There is a unique pure strategy Nash equilibrium, which is for each player to defect. This is a classic game in game theory and is an example of the tragedy of the commons where individuals acting rationally in their own self-interest prevent the group from obtaining its best interest. Specifically, if both players cooperate, then this is the socially optimal solution as they maximise the sum of their rewards, but defect is the dominant action for each player as it gives the highest reward irrespective of the other player's strategy. The payoff matrix is shown in Table 3.7 where longer sentences give smaller payoffs.

Table 3.7: Prisoner's dilemma payoff matrix, each entry shows (row player payoff, column player payoff) for the given row and column actions.

	D	C
D	1,1	4,0
C	0,4	3,3

Littman's Soccer

Littman's soccer game [Littman, 1994] is a two-player, zero-sum, stochastic game. It is played on a 4×5 grid with goals either side. Each player can move north, south, east, west or can stand still and must dribble the ball into its opponent's goal. Each game starts with the ball being given to a random player. At each step, players select actions simultaneously and these are executed in a random order, which adds non-determinism to the game. Scoring (being scored against) gets a reward of 1 (-1) and resets players to their initial positions. If a player tries to move to an occupied position, then the move fails and gives the ball to the opponent (if possible). Littman proposed this simplified soccer game to demonstrate his minimax-Q learning algorithm. Figure 3.2 shows the initial layout of the game, the large circles represent the players, the smaller circle represents the ball, and each player starts closer to its own goal.

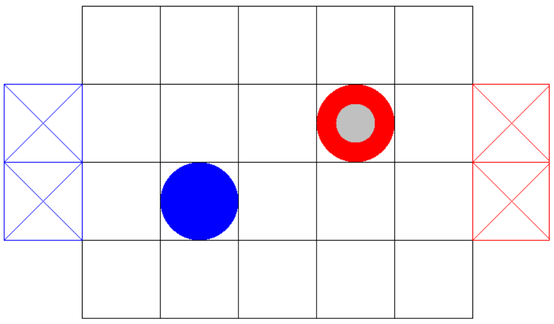


Figure 3.2: Littman's soccer game initial state, large circles are players, the small circle is the ball, and each player is closer to its own goal.

Die-Roll Poker

Die-roll poker is a two-player, zero-sum poker game where dice are used instead of cards. It was introduced by Lanctot et al., 2012 and proceeds as follows:

1. Each player antes one chip into the pot.
2. Each player rolls its first private six-sided die.
3. First public betting round occurs, each raise (max two) is two chips.
4. If no one folded, each player rolls its second private six-sided die.
5. Second public betting round occurs, each raise (max two) is four chips.
6. If no one folded, a showdown occurs, the highest dice sum wins the pot.

The game has imperfect information due to each player's die rolls initially being hidden from its opponent. If the game ends in a fold, then each player keeps its die rolls hidden from its opponent. If the game ends in a showdown, then each player reveals the sum of its die rolls to its opponent, but each individual die roll that constituted that sum is not revealed. For example, at a showdown a player might reveal to its opponent that the sum of its die rolls is three, but the opponent cannot tell if the player rolled either ⚀⚁ or ⚁⚀. The game was introduced by Lanctot et al., 2012 to help test their theory about regret bounds for using counterfactual regret minimisation in imperfect recall games. The structure of die rolls and betting rounds in the die-roll poker game tree are shown in Figure 3.3.

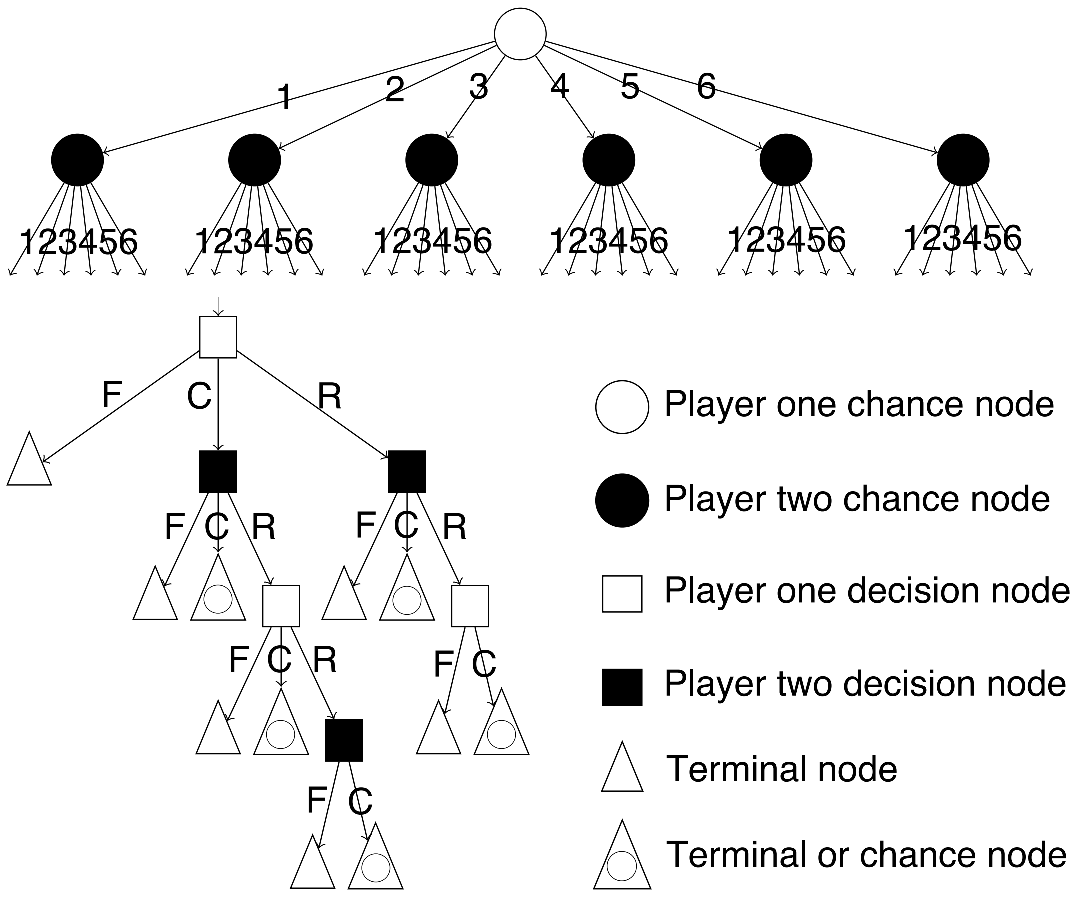


Figure 3.3: Die-roll poker game tree (die rolls and betting rounds). The top shows each player rolling a private six-sided die. The bottom left shows a betting round where "terminal or chance nodes" are terminal in the second betting round and chance in the first betting round. There are 1.12×10^5 nodes with 5.58×10^2 decision information sets per player.

Rhode Island Hold'em

Rhode Island hold'em is a two-player, zero-sum poker game, which like most poker games uses a standard fifty-two card deck. However, each player is only dealt one private card and only two public cards are dealt. The game was introduced by Shi and Littman, 2000 and proceeds as follows:

1. Each player antes five chips into the pot.
2. Each player is dealt one private card from a standard fifty-two card deck.
3. First public betting round occurs, each raise (max three) is ten chips.
4. If no one folded, the first public card, called the "flop", is dealt.
5. Second public betting round occurs, each raise (max three) is twenty chips.
6. If no one folded, the second public card, called the "turn", is dealt.
7. Third public betting round occurs, each raise (max three) is twenty chips.
8. If no one folded, a showdown occurs, the best three-card hand wins the pot.

The game has imperfect information due to each player's private card initially being hidden from its opponent. If the game ends in a fold, then each player keeps its private card hidden from its opponent. If the game ends in a showdown, then each player reveals its private card to its opponent. The game was introduced by Shi and Littman, 2000 to help test their abstraction methods. The structure of card deals and betting rounds in the Rhode Island hold'em game tree are shown in Figure 3.4. Three-card poker hand ranks are shown in Table 3.8.

Table 3.8: Three-card poker hand ranks. Ties are attempted to be broken in all cases by comparing the card with the highest rank, then the card with the second highest rank, then the card with the third highest rank.

Rank	Hand	Probability	Description	Example
1	Straight flush	0.0022	Three sequential cards sharing suit.	$Q\spadesuit K\spadesuit A\spadesuit$
2	Three of a kind	0.0024	Three cards sharing rank.	$7\clubsuit 7\heartsuit 7\diamondsuit$
3	Straight	0.0326	Three sequential cards.	$3\heartsuit 4\clubsuit 5\spadesuit$
4	Flush	0.0496	Three cards sharing suit.	$2\diamondsuit 7\diamondsuit K\diamondsuit$
5	Pair	0.1694	Two cards sharing rank.	$9\clubsuit Q\heartsuit Q\diamondsuit$
6	High card	0.7439	None of the above.	$4\spadesuit 10\spadesuit J\clubsuit$

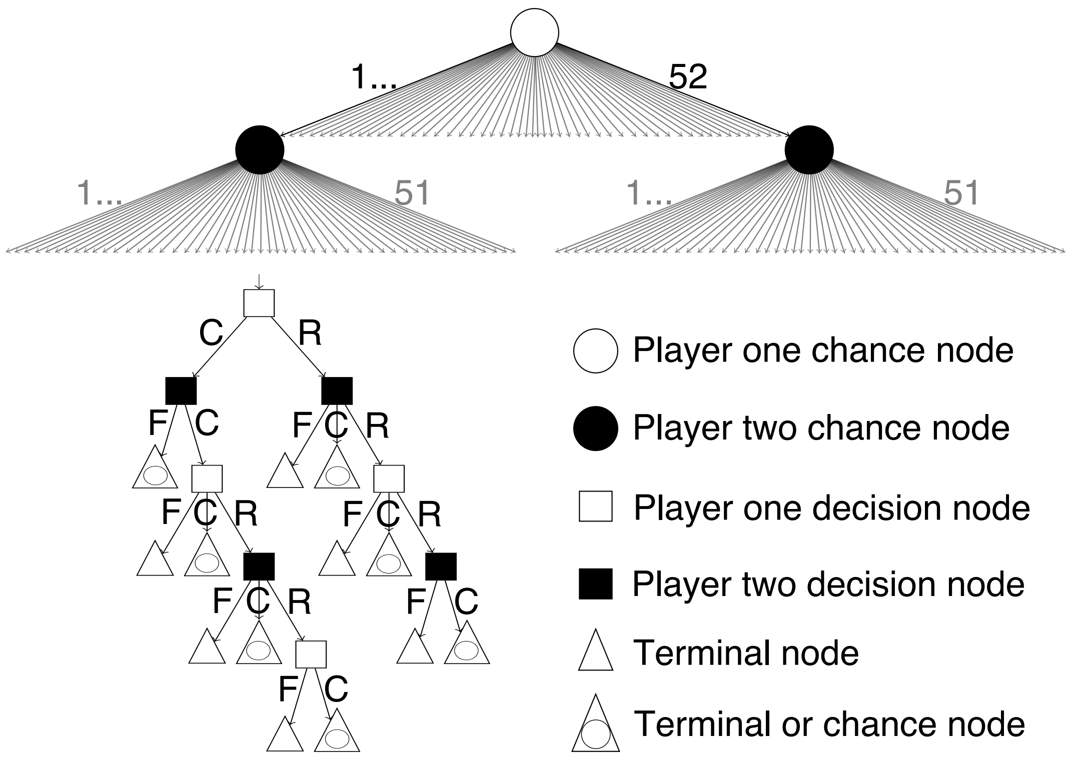


Figure 3.4: Rhode Island hold'em game tree (private card deals and betting rounds). The top shows each player being dealt its private card out of a standard fifty-two card deck. The bottom left shows a betting round where "terminal or chance nodes" are terminal in the third betting round and chance in the first and second betting rounds. Not shown are public chance node branches, with 50 branches from each "flop" chance node and 49 branches from each "turn" chance node. There are 6.71×10^9 nodes with 2.50×10^7 and 2.46×10^7 decision information sets for player one and two respectively.

3.1.8 Abstraction by Bucketing

Some games have large state spaces or numbers of states. For example, heads-up (two-player) limit Texas hold'em has about 1×10^{18} (quintillion) states [Zinkevich et al., 2008]. This can make them difficult to store in terms of memory and difficult to learn strategies for in terms of computation time required to consider all decisions. Practically only a limited number of states can be handled. One way to reduce the number of states is to combine states that are similar according to some metric. The experience in a combined state is then generalised over the states it represents. Examples of state space abstraction include coarse coding, tile coding, Kanerva coding, and the use of radial basis functions. A state space abstraction technique is used in Rhode Island hold'em called bucketing.

Bucketing groups information sets that are deemed to have similar values by some metric. The assumption is that if information sets are in the same bucket, with similar values, then the optimal behaviour will be similar for all of them and so instead of learning a behaviour for each of them, one behaviour is learnt for all of them. The reduction in the number of information sets to learn behaviours for makes the game smaller. Typically, the smaller a game is, the faster an agent can learn an effective strategy within that game. Bucketing is usually applied to poker-like games and this is the type of games this thesis uses it for. Most of the information on bucketing is based on that found in Johanson, 2007.

Expected Hand Strength

Expected hand strength is a metric designed for poker-like games and is defined according to Johanson, 2007 as the probability of a player's private cards winning at a showdown when the public cards that have not been dealt along with the opponents' private cards are "rolled out", which means they are sampled uniformly at random without replacement from the remaining possible cards. The procedure for calculating expected hand strength depends on the stage of the game. If the game is at a showdown, where you can see your opponents' cards or hidden information, then your expected hand strength is 1 if your hand wins, 0 if your hand draws and -1 if your hand loses. If the game is at a point where all private and public cards have been dealt, but the opponents' cards are unknown, then your expected hand strength is calculated by dealing all possible combinations of opponent cards, counting how many times your hand wins, draws and loses against them and calculating

$$E[HS] = \frac{\text{wins} + \text{draws}/2}{\text{wins} + \text{draws} + \text{losses}}. \tag{3.13}$$

Finally, if the hand is at a point where all private cards have been dealt, but only some public cards have been dealt and the opponents' cards are unknown, then your expected hand strength is calculated by dealing all possible combinations of public cards and averaging the expected hand strength of each combination. The expected hand strength squared, denoted by $E[HS]^2$, is simply the square of the expected hand strength. The purpose of squaring the expected hand strength is to give more weight to hands that, although initially weak, could become strong.

Percentile Bucketing

Percentile bucketing divides all n-card hands, where n is fixed, evenly between a set of buckets $B_n = \{b_1, b_2, \ldots, b_{|B_n|}\}$. For example, in the case of Rhode Island hold'em, a player's hand has one card pre-flop $n = 1$, two cards on the flop $n = 2$ and three cards on the turn $n = 3$. At each of these stages, the player's hand can be categorised as belonging to a bucket of hands with the same number of cards and a similar strength. Normally each bucket $b_m \in B_n$ divides hands into $|B_n|$ ranges along the chosen metric. For example, using expected hand strength, the first bucket $b_1 \in B_n$ would group hands with a value $0 \leq E[HS] < \frac{1}{|B_n|}$, the second bucket $b_2 \in B_n$ would group hands with a value $\frac{1}{|B_n|} \leq E[HS] < \frac{2}{|B_n|}, \ldots,$ the last bucket $b_{|B_n|} \in B_n$ would group hands with a value $\frac{|B_n|-1}{|B_n|} \leq E[HS] \leq 1$. However, a problem with this abstraction is that most hand strengths are mediocre with only a few being very good or very bad. This means that most hands end up in the middle buckets whilst the buckets near the start and the end are almost empty. The reason this is bad is because a bucket containing many hands will cause the agent to be unable to differentiate strategically between a lot of different situations and a bucket containing few hands will be almost not worth having in terms of reducing the size of the game. Percentile bucketing deals with this by ensuring that all buckets contain an approximately equal number of hands. The way it does this is by ordering the hands according to the chosen metric and then dividing the ordered hands evenly into the $|B_n|$ buckets.

Bucketed Rhode Island Hold'em

This is an abstraction of Rhode Island hold'em which reduces its information sets where players act from 2.50×10^7 for player one and 2.46×10^7 for player two to 2.52×10^3 each. This allows the agents to learn effective strategies within 1×10^5 games, which is the number of games chosen to evaluate agents over in the experiments in Chapter 5. Evidence for this is discussed in Section 5.6.1. The abstraction uses percentile bucketing based on expected hand strength squared. Expected hand strength is the probability of a player's private cards combined with the public cards winning against a uniform random draw of the opponent's private cards combined with the public cards. Expected hand strength squared is simply the square of the expected hand strength. This gives more weight to initially weak hands that could become strong such as straights or flushes. Percentile bucketing divides all n-card hands evenly between a set of buckets B_n, where in Rhode Island hold'em $n \in \{1, 2, 3\}$. For more information on percentile bucketing and expected hand strength see Section 3.1.8 as well as Johanson's MSc thesis [Johanson, 2007].

3.2 Learning in Games

3.2.1 Repeating Games

Repeating a game is usually essential for a player to learn an effective strategy. If a player has prior knowledge about the game, or the other players, or both, then it may already know an effective strategy. In this case learning may be unnecessary, for example, a player may know a single dominant pure strategy. However, in most cases, with or without prior knowledge, a player can learn to improve its strategy. The number of games required to learn an effective strategy generally depends on factors such as stochasticity in the game, payoff variance, the player's strategy, and the other players' strategies. Additionally learning time generally increases with the size of the strategy space, which depends on the number of states and actions per state. Ideally, a player wants to learn the optimal action in each state where it acts. In game theory, a repeated or iterated game usually refers to playing a stage game numerous times, where stage games are typically two-player, two-action, normal-form games such as the prisoner's dilemma. In these games, where each player plays one action per game, repetition fundamentally changes the interaction as it allows a strategy to depend on past actions.

3.2.2 Stochasticity in Games

In a game a stochastic event causes a state transition according to a random element. It can be due to a player's probabilistic decision or a rule in the game such as a die roll or card deal. A stochastic event that is not due to a normal player is usually attributed to a special “chance” or “nature” player who has no payoffs. This allows for a consistent view where any event, which is either deterministic or stochastic, is due to a player's decision. In extensive-form representation, it means that every node (state) is “owned” by a player who makes decisions at that point. In game theory, a stochastic game usually refers to the type of game defined in Section 3.1.3.

3.2.3 The Problem of Convergence

Crawford's puzzle is about the difficulty of converging to mixed strategy Nash equilibria in games. This thesis looks at this problem in Chapter 7. Also many of the agents used to develop, or used as comparisons to, the approaches in this thesis, tackle this problem. Understanding this problem will help towards understanding those agents. Crawford first raised the issue of non-convergence through a series of papers starting in 1974. He looked at an infinitely repeated two-player game with a unique mixed strategy Nash equilibrium. He gave each player highly natural adaptation rules. Specifically, each agent used gradient ascent to adjust its strategy to give itself higher payoffs. He discovered that these agents were unable to converge to the mixed strategy Nash equilibrium. He showed that this was true for zero-sum matrix games [Crawford, 1974], general-sum matrix games [Crawford, 1985] and evolutionary games [Crawford, 1989]. The result was surprising because the setting seemed highly favourable for convergence. This is because, amongst other things, agents were interacting repeatedly, the problem was relatively straightforward, there was good feedback, and the mixed strategy Nash equilibrium was unique. Crawford's puzzle is a challenge to find plausible adaptation rules for the players that will converge to a Nash equilibrium. Much of the literature on multi-agent learning is devoted to developing learning agents with adaptation rules that can solve Crawford's puzzle. On the one hand, the problem is usually relaxed such that an agent just has to converge to a Nash equilibrium in self-play. However, the progression in the literature seems to be towards making fewer assumptions about an agent's available information.

3.2.4 Modes of Convergence

Much of the literature in multi-agent learning and game-theory focusses on learning, or converging to, solution concepts such as Nash equilibria. Simultaneously co-adapting agents in an uncooperative setting may converge, or may show complex behaviour. A number of researchers have investigated this[1]. This section explains several definitions of convergence, and what it means in this context. This will also set up Chapter 7, which looks at self-play convergence.

Convergence can be defined as the property or manner of approaching a limit, such as a point, line, function, or value. There are many types of convergence of random variables. The convergence of a sequence of random variables to some limit random variable is known as stochastic convergence, which is the idea that the behaviour of a sequence of random events can sometimes be expected to stabilise if items far enough into the sequence are studied. Two common behaviours are firstly, that the sequence returns a constant value in the limit and secondly, that the sequence acts as if sampled from a fixed distribution in the limit.

Let $(X_1, X_2, \ldots X_n)$ and X be real-valued random variables defined on the same probability space $(\Omega, \mathcal{F}, P)$. In this probability space, Ω is the sample space, which is the set of all possible outcomes, $\mathcal{F} = \{\Omega_1, \Omega_2, \ldots, \Omega_{|\mathcal{F}|}\}$ is a set of events, where each event is a subset of outcomes $\Omega_i \subseteq \Omega$ for $1 \leq i \leq |\mathcal{F}|$, and P is a function mapping from events to probabilities $P : \mathcal{F} \to [0, 1]$.

Pointwise convergence

Pointwise convergence is the strongest type of convergence, and implies convergence with probability one, convergence in probability, and convergence in distribution. If $(X_1, X_2, \ldots X_n)$ pointwise converges to X, then

$$\lim_{n\to\infty} X_n(\omega) - X(\omega) = 0 \text{ for all } \omega \in \Omega. \tag{3.14}$$

[1]Abdallah and Lesser, 2008; Awheda and Schwartz, 2013; Banerjee and Peng, 2003; Bowling, 2005; Bowling and Veloso, 2002; Butterworth and Shapiro, 2009; Crawford, 1974, 1985, 1989; Galla and Farmer, 2013; Shapley, 1963; Singh, Kearns, and Mansour, 2000; Zhang and Lesser, 2010.

Convergence with probability one

Convergence with probability one, or almost sure convergence, or convergence everywhere, is the same as pointwise convergence, but excludes events with zero probability. This is the type of convergence that is used by the strong law of large numbers. Convergence with probability one implies convergence in probability, and convergence in distribution. If $(X_1, X_2, \dots X_n)$ converges with probability one to X, then

$$\lim_{n\to\infty} X_n(\omega) - X(\omega) = 0 \text{ for all } \omega \in \Omega' \text{ such that } \Pr(\Omega \setminus \Omega') = 0. \tag{3.15}$$

Convergence in the k-th Mean

Convergence in the k-th mean implies convergence in probability (by Markov's inequality, $\Pr(\{\omega \in \Omega : X(\omega) \geq a\}) \leq E(X)/a$, where $a > 0$) if $k \geq 1$, as well as convergence in any i-th mean if $1 \leq i < k$, and finally convergence in distribution. If $(X_1, X_2, \dots X_n)$ converges in the k-th mean to X, then

$$\lim_{n\to\infty} E(|X_n - X|^k) = 0, \tag{3.16}$$

where E is the expected value over Ω.

Convergence in Probability

Convergence in probability implies convergence in distribution. This is the type of convergence that is used by the weak law of large numbers. If $(X_1, X_2, \dots X_n)$ converges in probability to X, then

$$\lim_{n\to\infty} \Pr(|X_n(\omega) - X(\omega)| \geq \epsilon) = 0 \text{ for all } \omega \in \Omega \text{ and } \epsilon > 0. \tag{3.17}$$

Convergence in Distribution

Convergence in distribution, or convergence in law, is the weakest type of convergence considered. If $(X_1, X_2, \dots X_n)$ converges in distribution to X, then

$$\lim_{n\to\infty} F_{X_n}(x) - F_X(x) = 0 \text{ for all } x \in \mathbb{R} \text{ where } F_X \text{ is continuous,} \tag{3.18}$$

F_{X_n} and F_X are the cumulative distribution functions of the random variables X_n and X respectively and $F_X(x) = \Pr(\{\omega \in \Omega : X(\omega) \leq x\})$. This means that the probability of the value of X_n being within a certain range becomes approximately equal to the probability that the value of X is within that range as n tends to infinity.

3.2.5 Convergence in Game Theory

Convergence in game theory is usually concerned with the convergence of an agent's strategy, or payoffs, or both. In particular, the convergence of an agent's strategy and expected payoffs to a best-response strategy and its expected payoffs, or the convergence of all agents' strategies and expected payoffs to a Nash equilibrium and its expected payoffs. In general-sum games, there could be multiple Nash equilibria, and some might be better than others by, for example, having strictly higher expected payoffs for all agents. Ideally, if all agents converge to a Nash equilibrium, then they should converge to the best (Pareto optimal) Nash equilibrium. Following this is an explanation of what it means for an agent's strategy or payoffs to converge.

Convergence of an Agent's Strategy

Convergence of an agent's strategy to some other strategy means that, in the limit, its strategy will behave exactly the same as the converged-to strategy. As an agent plays a (possibly repeated) game, the result will be a sequence of random variables for each of its information sets where it acts. If its strategy converges, then for each information set where it acts, the corresponding sequence of random variables will pointwise converge to the random variable in the converged-to strategy for that information set as the number of random variables in the sequence approaches infinity. Formally this means that

$$\lim_{t \to \infty} \sigma^t_{\text{pla}}(I, a) - \sigma^*_{\text{pla}}(I, a) = 0 \text{ for all } I \in \mathcal{I}_{\text{pla}} \text{ and } a \in A(I) \tag{3.19}$$

where $\sigma^t_{\text{pla}}(I)$ and $\sigma^*_{\text{pla}}(I)$ are random variables for the agent's strategy at time t and for the converged-to strategy respectively both at the agent's information set $I \in \mathcal{I}_{\text{pla}}$ and $A(I)$ is the set of actions (sample space) at information set $I \in \mathcal{I}_{\text{pla}}$.

Convergence of an Agent's Empirical Distribution of Plays

Convergence of an agent's empirical distribution of plays to some strategy is the same as convergence of an agent's strategy to that strategy except its empirical distribution of plays is used instead. An agent's empirical distribution of plays defines the probability distribution over the outcomes of each random variable at each information set where the agent acts as the number of times each of those outcomes occur divided by the total number of outcomes i.e.

$$\pi^t_{\text{pla}}(I,a) = \frac{M_v((I,a))}{\sum_{a\in A(I)} M_v((I,a))} = \frac{M_v((I,a))}{M_v(I)} \text{ for all } I \in \mathcal{I}_{\text{pla}} \text{ and } a \in A(I), \tag{3.20}$$

where π^t_{pla} is the agent's empirical distribution of plays at time t, $M_v : \mathcal{I}_{\text{pla}} \to \mathbb{R}$ is a map from the agent's information sets to real numbers such that given an information set $I \in \mathcal{I}_{\text{pla}}$ then $M_v(I)$ will return how many times that information set has been visited. This means $M_v((I,a))$ is the number of times information set $(I,a) \in \mathcal{I}_{\text{pla}}$ has been visited or equivalently, how many times action $a \in A(I)$ has been played in information set $I \in \mathcal{I}_{\text{pla}}$. If an agent's empirical distribution of plays converges, then

$$\lim_{t\to\infty} \pi^t_{\text{pla}}(I,a) - \sigma^*_{\text{pla}}(I,a) = 0 \text{ for all } I \in \mathcal{I}_{\text{pla}} \text{ and } a \in A(I) \tag{3.21}$$

where $\pi^t_{\text{pla}}(I)$ and $\sigma^*_{\text{pla}}(I)$ are random variables for the agent's empirical distribution of plays at time t and for the converged-to strategy respectively both at the agent's information set $I \in \mathcal{I}_{\text{pla}}$ and $A(I)$ is the set of actions or the sample space at information set $I \in \mathcal{I}_{\text{pla}}$.

Convergence of an Agent's Expected Reward

Convergence of the expected reward of an agent's strategy to some real value means that, in the limit, the expected reward of its strategy will always be that reward. In general, this does not mean that the agent's strategy converges. For example, in rock-paper-scissors, the agent's expected reward could converge to 1 against an opponent who plays (Rock, Paper, Scissors) in a loop if it plays (Paper, Scissors, Rock) in a loop. If the expected reward of an agent's strategy converges, then

$$\lim_{t\to\infty} \overline{u}_{\text{pla}}(\sigma^t) - x = 0, \tag{3.22}$$

where $\overline{u}_{\text{pla}}(\sigma^t))$ is the expected reward or utility that the agent receives given the strategy profile at time t, and $x \in \mathbb{R}$ is the converged to expected reward.

Convergence of an Agent's Average Reward

Convergence of an agent's average reward to some real value means that, in the limit, the average of its rewards equals the converged-to value. If an agent's average reward converges, then

$$\lim_{t \to \infty} \left[\frac{1}{t} \sum_{i=1}^{t} u_{\text{pla}}(h_t) \right] - x = 0 \tag{3.23}$$

where $u_{\text{pla}}(h_t)$ is the reward or utility that the agent receives from reaching the history $h_t \in H$ at time t and $x \in \mathbb{R}$ is converged to average reward.

Strong and Weak Convergence

Convergence of an agent's strategy is the strongest notion of convergence because it implies the three other notions of convergence (i.e. convergence of an agent's empirical distributions of plays, convergence of the expected reward of its strategy, and convergence of its average reward). For this reason, when an agent's strategy is said to "strongly" converge it usually means that its actual strategy converges, and when an agent's strategy is said to "weakly" converge it usually means that its empirical distribution of plays converges. Strictly speaking, the strategy convergence definition requires pointwise convergence at all the agent's information sets where it acts. However, this could be weakened such that it only pointwise converges at information sets where it acts that are reachable. This would mean that information sets that will not be encountered either due to the agent's strategy, or an opponent's strategy, could be ignored.

3.2.6 Machine Learning

Machine learning is the concept of a system learning automatically from data to perform well in a given task. Machine learning algorithms can be distinguished by the way they learn from experience. A supervised learning algorithm is trained through direct feedback, in that for each decision, it is told the correct answer. In other words, it learns a function given input-output examples. In contrast, an unsupervised learning algorithm has no direct feedback, and is not told the correct answer. Instead, it is just given the inputs and attempts to model them by, for example, clustering. A semi-supervised learning algorithm is somewhere in-between, it is given some, possibly indirect, input-output examples.

The approaches in this thesis use reinforcement learning and no-regret learning algorithms, which are semi-supervised learning algorithms. In general, acting in an environment changes its state and generates feedback. These algorithms learn action values from this feedback. On the surface this looks like supervised learning with decisions (actions) being associated directly with feedback (rewards). The difference is that the feedback may be delayed, a feedback in one state may be for a set actions taken many states earlier. For example, in chess (or any game really), the feedback could be 1 for winning the game, 0 for a draw and -1 for a loss with no indication as to what actions are responsible for the feedback.

One advantage of machine learning is that it may discover superior solutions than can be found manually. For example, TD-Gammon, Tesauro's backgammon playing program, achieved a top-10 human-level performance by learning in self-play using temporal difference (TD) learning without any prior knowledge [Tesauro, 1992, 1994, 1995, 1998, 2002]. It even changed the way human grandmasters played from certain positions [Tesauro, 1995, 2002]. A second advantage is that it may find approximate solutions when analytical solutions are infeasible. A third advantage is that it can generally handle lots of data, which may be difficult to process manually. This is usually the case in game-playing, where although individual games may be small, they can be repeated many times generating lots of states, actions, and rewards.

We want agents to learn effective strategies, which requires them to accurately value their possible actions in each state, and to do this they must deal with two main problems. The first problem is the exploration versus exploitation trade-off, which requires an agent to choose to either play its best known actions (exploit), or to play other actions that may give higher rewards (explore). The second problem is the credit assignment problem, which requires an agent to choose how eligible each prior action is for a feedback signal. If these problems are dealt with, then reinforcement learning and no-regret learning may be viable options over supervised learning where we may not have the resources (e.g. time or knowledge) to provide agents with lots of exact training examples.

3.2.7 Reinforcement Learning

The question an agent faces when put into an environment in a particular state is: What is the right action? Its chosen action may have no effect on the environment, or it may change the state of the environment. Unlike supervised learning agents, who are given the correct choices after choosing actions, reinforcement learning agents are only given indications as to the quality of their actions. Thus, reinforcement learning agents are not entirely unsupervised because they receive some feedback in the form of rewards for reaching certain states. This puts reinforcement learning between supervised and unsupervised learning in the taxonomy of machine learning algorithms, but much closer to the latter. It learns how to act to maximise some notion of reward through observations of the world.

Reinforcement learning has become a standard approach for a learning agent to discover optimal actions given perceptions of environment states. For a good introduction to reinforcement learning, including examples of its applications see Sutton and Barto, 1998. A reinforcement learning problem specifies rewards for reaching goals, but not how to reach them. This makes it a useful framework for problems where we know the goals, and their importance, but not the actions required to reach them, or if we think a reinforcement learning algorithm can find better actions to reach them. Another advantage of reinforcement learning is that it is a naturally intuitive way to tackle a problem as it is similar to how humans and animals learn to solve problems. If we have a task, then we are usually aware of what we want to achieve but not, at least initially, how to achieve it. Better decisions are made in both cases by selecting actions and observing their results. Future decisions at the same or similar situations are then guided by experience.

The feedback for taking an action, which is essential for reinforcement learning, is usually a numerical reward signal received when entering an environment state. Higher reward values indicate more preferred states. A rational reinforcement learning agent aims to find the action, or set of actions, for each state that yield the largest expected reward within a given time frame. This must be done through some form of trial and error that addresses the exploration versus exploitation and credit assignment problems. Traditionally, reinforcement learning focusses on single-agent learning and ignores other agents. The only concern being how to choose the agent's actions to maximise its rewards, even though it is likely that in a multi-agent system other agents' actions are affecting those rewards. Specifically, each agent has its own desired environment states that it would be trying to reach through its actions. By ignoring other agents, it would be harder to understand and control changes in the environment.

3.2.8 Reinforcement Learning Algorithms

The reinforcement learning algorithms in this section are designed to enable an agent to learn an optimal strategy, which maximises its rewards. These algorithms deal with both the exploration vs exploitation and credit assignment problems. The former is where the algorithm must choose how often to explore taking actions whose values are either unknown or estimated as low but may have changed, as well as how often to exploit taking actions whose values are estimated to be high. The latter is where the algorithm must choose how to distribute a reward across all the actions taken before receiving it.

The first algorithm, Quality-Learning (Q-Learning) proposed by Watkins, 1989, is designed to learn the maximum discounted future reward of each action in a finite and stationary MDP. These values can be used to create an optimal greedy strategy, which simply takes the action with the highest value in each state. Q-Learning can be used in a multi-player game in self-play and although it loses its convergence guarantees, it generally works well. However, players' strategies will never converge to a mixed strategy Nash equilibrium because the optimal greedy strategy suggested by Q-Learning is deterministic.

Ideally we want players' strategies to converge to a mixed strategy Nash equilibrium if that represents a desired solution to the game. Each of the other algorithms in this section is designed to tackle this problem such that it can converge in self-play to a greedy mixed strategy such as a mixed strategy Nash equilibrium. The differences between them are the range of games that they converge to a Nash equilibrium in self-play within as well as their convergence speeds. In general, Policy Gradient Ascent with Approximate Policy Prediction (PGA-APP) converges in more games and at a faster rate than Weighted Policy Learner (WPL), which in turn converges in more games and at a faster rate then Win or Learn Fast Policy Hill Climbing (WoLF-PHC). One similarity is that they all use Q-Learning to estimate action-values. They each differ in how they use their action-values to derive estimates of optimal and possibly mixed strategies.

In short, WoLF-PHC converges by using a variable learning rate to update its strategy, which is set high if it is losing and low if it is winning. WPL also converges by using a variable learning rate to update its strategy, which is set to be proportional to the distance between its strategy and the simplex boundary that its strategy is being updated towards. PGA-APP converges by updating its strategy according to its opponent's anticipated strategy.

Quality-Learning (Q-Learning)

Q-Learning [Watkins, 1989] estimates the optimal action-value function independently of the strategy being used. The optimal action-value function estimates the maximum discounted expected reward of taking action a in state s and following the optimal strategy thereafter. Each (q-)value associated with a state-action pair is updated via the following equation

$$Q^{t+1}\left(s^t,a^t\right) \leftarrow \left(1-\alpha^t(s^t,a^t)\right) Q^t\left(s^t,a^t\right)+\alpha^t(s^t,a^t)\left[r^{t+1}+\gamma \max_{a^{t+1}} Q^t\left(s^{t+1},a^{t+1}\right)\right] \tag{3.24}$$

where Q^t is the estimated action-value function at time t, s^t is the state at time t, a^t is the action at time t, $0 \leq \alpha^t(s^t,a^t) \leq 1$ is the learning rate for state s^t and action a^t at time t, r^{t+1} is the reward at time $t+1$ after taking action a^t in state s^t at time t, $0 \leq \gamma \leq 1$ is the discount factor, and $Q^t\left(s^t,a^t\right)$ returns the time t estimated maximum discounted expected reward for action a^t in state s^t.

The learning rate α determines how quickly new information overrides old information. If $\alpha = 0$, then the algorithm will learn nothing and keep its initial action-values. If $\alpha = 1$, then the algorithm will remember nothing and always estimate an action-value using only its immediate reward and the estimated maximum discounted expected reward. The discount factor γ determines how much future rewards are valued and can be seen as the amount of implicit lookahead. If $\gamma = 0$, then the algorithm will be short-sighted and only consider immediate rewards. As γ approaches 1, the algorithm will value future action-values more and more. If $\gamma \geq 1$, then the action-values may diverge.

If the estimated action-values accurately represent maximum discounted expected rewards, then the optimal strategy is to select the action with the highest value in each state. Q-Learning has a variety of benefits. Firstly, it does not need a model of the environment in that it does not need to learn the state transition or reward functions. Secondly, it is an off-policy learning technique, which means that any strategy can be used and it is still guaranteed converge to the maximum discounted expected rewards as long as some conditions are met.

According to Watkins, 1989 and Szepesvári, 1998 Q-Learning is guaranteed to converge to the maximum discounted expected rewards with probability one under the following conditions. Firstly, the environment must be a finite and stationary MDP. Secondly, each action in each state must always have some non-zero probability of being taken. Thirdly, the learning rate must be decreased with an appropriate schedule. Specifically, for each state-action pair, the sum of the learning rates for that state-action pair must asymptomatically converge to infinity $\sum_{t=1}^{\infty} \alpha^t(s,a) = \infty$, and the sum of the squares of the learning rates for that state-action pair must asymptomatically converge to less than infinity $\sum_{t=1}^{\infty} \alpha^t(s,a)^2 < \infty$. Fourthly, the rewards must be bounded. Finally, action-values must be stored perfectly (i.e. without function approximation).

Q-Learning on its own is not an agent, but a method of estimating the maximum discounted expected rewards of state-action pairs. A common approach to create an agent based on it, which is adopted in this thesis, is to almost always select the action in a state with the maximum estimate. Sometimes multiple actions share the same maximum estimate and to handle this one of these can be selected uniformly at random. For simplicity, this detail is avoided in the following algorithms. This Q-Learning algorithm is shown in Algorithm 1.

Algorithm 1 Quality-Learning (Q-Learning)

```
Require: Set of all states $S$, set of all actions $A$, learning rate $0 \le \alpha \le 1$,
    discount factor $0 \le \gamma \le 1$.
1: Initialise each action-value $Q^0(s,a) \leftarrow 0$ and each action probability $\sigma(s,a) \leftarrow$
   $1/|\mathcal{A}|$, for all states $s \in S$ and actions $a \in A$.
2: From state $s^t \in S$ at time $t$ select action $a^t \in A$ at time $t$ according to
   probability $\sigma(s^t,a^t)$ with some exploration.
3: Observe reward $r^{t+1}$ and next state $s^{t+1}$.
4: Update $Q^{t+1}(s^t,a^t) \leftarrow (1-\alpha) Q^t(s^t,a^t) + \alpha [r + \gamma \max_{a^{t+1}} Q^t(s^{t+1},a^{t+1})]$.
5: Update the probability of each action $a$ in state $s^t$
6: for all $a \in A$ do
7:     $\sigma(s^t,a) \leftarrow \begin{cases} 1 & \text{if } a = \arg\max_{a'} Q^{t+1}(s^t,a'), \\ 0 & \text{otherwise.} \end{cases}$
8: end for
9: Repeat from 2.
```

Win or Learn Fast Policy Hill Climbing

Win or Learn Fast (WoLF) [Bowling and Veloso, 2001, 2002] is a principle for varying an agent's learning rate to improve its performance. The idea is to decrease its strategy learning rate when it is winning, which increases the time spent playing a winning strategy and exploiting the opponent, and to increase its strategy learning rate when it is losing, which decreases the time spent playing a losing strategy and the opponent exploiting it. An agent is defined to be winning if the maximum discounted expected reward of its current strategy exceeds the maximum discounted expected reward of some safe strategy such as a Nash equilibrium strategy.

WoLF-PHC is a multi-agent reinforcement learning algorithm that implements the WoLF principle. For WoLF-PHC its safe strategy is its average strategy, which is defined as

$$\bar{\sigma}(s^t,a) \leftarrow \bar{\sigma}(s^t,a) + \frac{1}{C(s^t)}(\sigma(s^t,a) - \bar{\sigma}(s^t,a)) \text{ where } \bar{\sigma}(s^0,a) \leftarrow 0, \tag{3.25}$$

$\bar{\sigma}(s^t,a)$ is the average probability of playing action a in state s at time t, $C(s^t)$ is the number of times state s has been visited by time t and $\sigma(s^t,a)$ is the probability of playing action a in state s at time t. WoLF-PHC estimates the maximum discounted expected rewards of its actions in a state using Q-Learning. Thus, it estimates the maximum discounted expected reward of its strategy in a state as $\sum_a \sigma(s^t,a)Q^t(s^t,a)$. The strategy learning rate for a particular state δ is then set to

$$\delta = \begin{cases} \delta_w & \text{if } \sum_a \sigma(s^t,a)Q^t(s^t,a) > \sum_a \bar{\sigma}(s^t,a)Q^t(s^t,a), \\ \delta_l & \text{otherwise,} \end{cases} \tag{3.26}$$

where δ_w and δ_l are the winning and losing strategy learning rates respectively. The strategy is updated using the strategy learning rate such that the probability of the action with the estimated maximum discounted expected reward is increased and the probabilities of the other actions are decreased as

$$\sigma(s^t,a) \leftarrow \sigma(s^t,a) + \begin{cases} \delta & \text{if } a = \arg\max_{a'} Q^t(s^t,a'), \\ \frac{-\delta}{|\mathcal{A}_i|-1} & \text{otherwise.} \end{cases}. \tag{3.27}$$

WoLF-PHC is an approximate version of Win or Learn Fast Iterated Gradient

Ascent (WoLF-IGA). Bowling and Veloso, 2001, 2002 proved that the latter converges to a best-response strategy against stationary opponents and to a mixed strategy Nash equilibrium in self-play in two-player, two-action, general-sum repeated normal-form games. For the former, they showed that this convergence is achieved in this class of games as well as in two stochastic games (gridworld and Littman's soccer) and Jordan's game. The WoLF-PHC algorithm is shown in Algorithm 2.

Algorithm 2 Win or Learn Fast Policy Hill Climbing (WoLF-PHC)

```
Require: Set of all states $S$, set of all actions $A$, learning rate $0 \leq \alpha \leq 1$, discount factor $0 \leq \gamma \leq 1$, winning learning rate $\delta_w$ and losing learning rate $\delta_l$ where $\delta_l > \delta_w$.
1: Initialise each action-value $Q^0(s,a) \leftarrow 0$, each action probability $\sigma(s,a) \leftarrow 1/|\mathcal{A}|$, each average action probability $\bar{\sigma}(s,a) \leftarrow 0$ and the number of visits to each state $C(s) \leftarrow 0$, for all states $s \in S$ and actions $a \in A$.
2: From state $s^t \in S$ at time $t$ select action $a^t \in A$ at time $t$ according to probability $\sigma(s^t,a^t)$ with some exploration.
3: Observe reward $r$ and next state $s^{t+1}$.
4: Update $Q^{t+1}(s^t,a^t) \leftarrow (1-\alpha)\,Q^t(s^t,a^t) + \alpha\left[r + \gamma \max_{a^{t+1}} Q^t(s^{t+1},a^{t+1})\right]$.
5: Increment the number of visits to the current state $C(s^t) \leftarrow C(s^t)+1$.
6: Update the average probability of each action $a$ in state $s^t$
7: for all $a \in A$ do
8:     $\bar{\sigma}(s^t,a) \leftarrow \bar{\sigma}(s^t,a) + \frac{1}{C(s^t)}(\sigma(s^t,a) - \bar{\sigma}(s^t,a))$.
9: end for
10: Set the learning rate $\delta = \begin{cases} \delta_w & \text{if } \sum_a \sigma(s^t,a)Q^{t+1}(s^t,a) > \sum_a \bar{\sigma}(s^t,a)Q^{t+1}(s^t,a), \\ \delta_l & \text{otherwise.} \end{cases}$
11: Update the probability of each action $a$ in state $s^t$
12: for all $a \in A$ do
13:     $\sigma(s^t,a) \leftarrow \sigma(s^t,a) + \begin{cases} \delta & \text{if } a = \arg\max_{a'} Q^{t+1}(s^t,a'), \\ \frac{-\delta}{|\mathcal{A}_i|-1} & \text{otherwise.} \end{cases}$
14: end for
15: $\sigma(s^t) \leftarrow \text{project}(\sigma(s^t))$.    ▷ Ensure legal distribution (within simplex).
16: Repeat from 2.
```

Weighted Policy Learner

Weighted Policy Learner (WPL) [Abdallah and Lesser, 2008] is a multi-agent reinforcement learning algorithm. The idea is to speed up learning for an action when the gradient of the agent's value function with respect to that action changes direction and to slow down learning when the direction remains unchanged. WPL estimates the gradient of the agent's value function with respect to an action as

$$\Delta(s^t, a) \leftarrow Q^t(s^t, a) - \frac{\sum_{a \in A(s^t)} Q^t(s^t, a)}{|A(s^t)|} \tag{3.28}$$

where $\Delta(s^t, a)$ is the estimate of the gradient of the agent's value function with respect to action a in state s at time t, $Q^t(s^t, a)$ is the Q-Learning estimate of the maximum expected reward of action a in state s at time t and $A(s^t)$ is the set of available actions in state s at time t.

If the estimated value of an action is increasing, then its estimated gradient will be positive and its gradient is weighted by one minus the action probability. Otherwise, the estimated value of the action is decreasing, so its estimated gradient will be zero or negative and its gradient is weighted by the probability of the action. This gives

$$\Delta(s^t, a) \leftarrow \begin{cases} \Delta(s^t, a)(1 - \sigma(s^t, a)) & \text{if } \Delta(s^t, a) > 0, \\ \Delta(s^t, a)\sigma(s^t, a) & \text{otherwise,} \end{cases} \tag{3.29}$$

where $\sigma(s^t, a)$ is the probability of playing action a in state s at time t. An action probability is then updated using its weighted gradient as

$$\sigma(s^t, a) \leftarrow \sigma(s^t, a) + \eta \Delta(s^t, a) \tag{3.30}$$

where $0 \leq \eta \leq 1$ is a step size. This means that an action probability is increased the most when its gradient is positive and its probability is low, and an action probability is decreased the most when its gradient is negative and its probability is high. This makes sense because if an agent is not playing a winning action often, then it wants to quickly learn to play it more often and conversely if an agent is playing a losing action often, then it wants to quickly learn to play it less often.

One effect of these dynamics is that as an agent's probability distribution over its actions approaches the boundary of the simplex of legal probability distributions, it slows down. Therefore, probability distributions within the simplex will be approached faster than those on the boundary. This helps to prevent cycles and encourage convergence. WPL is (informally) proven to converge in self-play in two-player, two-action, general-sum repeated matrix games. It was shown experimentally to give faster convergence than WoLF-PHC to mixed strategy Nash equilibria in self-play in these games as well as in the challenging two-player, three-action Shapley game. The WPL algorithm is shown in Algorithm 3.

Algorithm 3 Weighted Policy Learner (WPL)

```
Require: Set of all states $S$, set of all actions $A$, learning rate $0 \leq \alpha \leq 1$,
    discount factor $0 \leq \gamma \leq 1$ and step-size $0 \leq \eta \leq 1$.
 1: Initialise each action-value $Q^0(s,a) \leftarrow 0$, each action probability $\sigma(s,a) \leftarrow \frac{1}{|\mathcal{A}_i|}$ and the value of each state $V(s) \leftarrow 0$, for all states $s \in S$ and actions $a \in A$.
 2: From state $s^t$ at time $t$ select action $a^t$ at time $t$ according to probability $\sigma(s^t, a^t)$ with some exploration.
 3: Observe reward $r$ and next state $s^{t+1}$.
 4: Update $Q^{t+1}(s^t, a^t) \leftarrow (1-\alpha) Q^t(s^t, a^t) + \alpha [r + \gamma \max_{a^{t+1}} Q^t(s^{t+1}, a^{t+1})]$.
 5: Update the value of state $s^t$, $V(s^t) \leftarrow \frac{\sum_{a \in A(s^t)} Q^{t+1}(s^t,a)}{|A(s^t)|}$.
    Update the probability of each action $a$ in state $s^t$
 6: for all $a \in A(s^t)$ do
 7:     $\Delta(s^t, a) \leftarrow Q^{t+1}(s^t, a) - V(s^t)$,            ▷ Gradient estimate.
 8:     if $\Delta(s^t, a) > 0$ then    ▷ +ve gradient and so decrease as approaching 1.
 9:         $\Delta(s^t, a) \leftarrow \Delta(s^t, a)(1 - \sigma(s^t, a))$,
10:     else                        ▷ -ve gradient and so decrease as approaching 0.
11:         $\Delta(s^t, a) \leftarrow \Delta(s^t, a)\sigma(s^t, a)$,
12:     end if
13:     $\sigma(s^t, a) \leftarrow \sigma(s^t, a) + \eta\Delta(s^t, a)$.
14: end for
15: $\sigma(s^t) \leftarrow$ project($\sigma(s^t)$).        ▷ Ensure legal distribution (within simplex).
16: Repeat from 2.
```

Policy Gradient Ascent with Approximate Policy Prediction

Policy Gradient Ascent with Approximate Policy Prediction (PGA-APP) [Zhang and Lesser, 2010] is a multi-agent reinforcement learning algorithm. The idea is that the agent adjusts its strategy based on the anticipated future strategies of the other players, instead of their current ones. Similarly to WoLF-PHC, PGA-APP estimates the maximum expected discounted rewards of its actions in a state using Q-Learning and thus estimates the maximum expected discounted reward of its strategy in a state as $\sum_{a \in A(s^t)} \sigma(s^t, a) Q^t(s^t, a)$. It uses this to estimate the gradient of the agent's value function with respect to an action as

$$\hat{\delta}(s^t, a) \leftarrow \begin{cases} Q^t(s^t, a) - \sum_{a \in A(s^t)} \sigma(s^t, a) Q^t(s^t, a) & \text{if } \sigma(s^t, a) = 1, \\ \frac{Q^t(s^t,a) - \sum_{a \in A(s^t)} \sigma(s^t,a) Q^t(s^t,a)}{1 - \sigma(s^t,a)} & \text{otherwise,} \end{cases} \tag{3.31}$$

where $\hat{\delta}(s^t, a)$ is the estimate of the gradient of the agent's value function with respect to action a in state s at time t, $Q^t(s^t, a)$ is the Q-Learning estimate of the maximum expected discounted reward of action a in state s at time t, $A(s^t)$ is the set of available actions in state s at time t and $\sigma(s^t, a)$ is the probability of playing action a in state s at time t. This is then modified to account for the anticipated future strategies of the other players as

$$\delta(s^t, a) \leftarrow \hat{\delta}(s^t, a) - \lambda|\hat{\delta}(s^t, a)|\sigma(s^t, a), \tag{3.32}$$

where λ is the derivative prediction length $0 \leq \lambda \leq 1$. An action probability is then updated using its modified gradient as

$$\sigma(s^t, a) \leftarrow \sigma(s^t, a) + \eta\delta(s^t, a), \tag{3.33}$$

where $0 \leq \eta \leq 1$ is a step size. This means that as an action probability increases, the magnitude of its gradient also increases. The gradient is modified by decreasing it to take into account the opponent's anticipated strategy. If the gradient is positive, then this modification will decrease its magnitude making the agent learn more slowly. If the gradient is negative, then this modification will increase its magnitude making the agent learn more quickly. Although the dynamics differ, the principle of learning quickly when losing and slowly when winning is the same in WoLF-PHC, WPL and PGA-APP. The underlying principle, which PGA-APP approximates, is proven to converge to mixed strategy Nash equilibria in self-play in two-player, two-action repeated matrix games. PGA-APP was shown experimentally to give faster convergence than WoLF-PHC and WPL to mixed strategy Nash equilibria in self-play in general-sum games. The PGA-APP algorithm is shown in Algorithm 4.

Algorithm 4 Policy Gradient Ascent with Approximate Policy Prediction (PGA-APP)

```
Require: Set of all states $S$, set of all actions $A$, learning rate $0 \leq \alpha \leq 1$,
    discount factor $0 \leq \gamma \leq 1$, derivative prediction length $0 \leq \lambda \leq 1$ and
    step-size $0 \leq \eta \leq 1$.
 1: Initialise each action-value $Q^0(s,a) \leftarrow 0$, each action probability $\sigma(s,a) \leftarrow \frac{1}{|\mathcal{A}|}$
    and the value of each state $V(s) \leftarrow 0$, for all states $s \in S$ and actions $a \in A$.
 2: From state $s^t \in S$ at time $t$ select action $a^t \in A$ at time $t$ according to
    probability $\sigma(s^t, a^t)$ with some exploration.
 3: Observe reward $r$ and next state $s^{t+1}$.
 4: Update $Q^{t+1}(s^t, a^t) \leftarrow (1-\alpha)\, Q^t(s^t, a^t) + \alpha\, [r + \gamma \max_{a^{t+1}} Q^t(s^{t+1}, a^{t+1})]$.
 5: Update the value of state $s^t$, $V(s^t) \leftarrow \sum_{a \in A} \sigma(s^t, a) Q^{t+1}(s^t, a)$.
 6: Update the probability of each action $a$ in state $s^t$
 7: for all $a \in A$ do
 8:     if $\sigma(s^t, a) = 1$ then
 9:         $\hat{\delta}(s^t, a) \leftarrow Q^{t+1}(s^t, a) - V(s)$,
10:     else
11:         $\hat{\delta}(s^t, a) \leftarrow \frac{Q^{t+1}(s^t,a) - V(s)}{(1-\sigma(s^t,a))}$,
12:     end if
13:     $\delta(s^t, a) \leftarrow \hat{\delta}(s^t, a) - \lambda \left|\hat{\delta}(s^t, a)\right| \sigma(s^t, a)$,
14:     $\sigma(s^t, a) \leftarrow \sigma(s^t, a) + \eta \delta(s^t, a)$.
15: end for
16: $\sigma(s^t) \leftarrow \text{project}(\sigma(s^t))$.        ▷ Ensure legal distribution (within simplex).
17: Repeat from 2.
```

3.2.9 Counterfactual Regret Minimisation

This section explains Counterfactual Regret Minimisation (CFR) because it is used in this thesis to update agents' strategies in Chapter 5 (including the strategy of the proposed approach in that chapter). CFR is a state-of-the-art algorithm that, in self-play, computes an approximate Nash equilibrium in two-player, zero-sum, imperfect information games. It works by minimising counterfactual regret in self-play, which Zinkevich et al., 2008 proved minimises overall regret, causing the average strategy profile to approach a Nash equilibrium.

An agent's counterfactual regret for not playing an action is the difference between its expected reward for playing that action and its expected reward for playing its strategy, weighted by the probability of reaching the information set where it acts if the probability of each of its actions leading to that information set were set to one. Thus, this probability is the product of the probabilities of the prior opponents' actions. This can be thought of as "counterfactual" because it is the agent's expected reward for reaching one of its information sets where it acts *if it had tried to do so.* Before minimising an agent's counterfactual regret, it first calculates the agent's counterfactual regret for not playing each of its actions at each of its information sets where it acts. If an agent has a high counterfactual regret for one of its actions, then in expectation, it would have received a higher cumulative reward if it had played that action more often, and so the algorithm increases its probability of playing it. Specifically, if an action's cumulative counterfactual regret is positive, then CFR sets its probability equal to it, otherwise CFR sets its probability to zero, then it normalises the action probabilities at each information set.

It is important to note that during operation CFR does not actually play games. For each iteration, for an agent who uses it, it calculates all the agent's counterfactual regrets, updates all the agent's cumulative counterfactual regrets, and updates the agent's strategy using regret matching. It can do this because it knows the game tree and the agents' strategies. When CFR is used in self-play, it just means that each agent's strategy is updated using it, but again they are not actually playing games. Moreover, CFR is completely deterministic and does not sample any values. Zinkevich et al., 2008 defined a sampling version of CFR along with the original non-sampling version, but the sampling version was later recognised by Lanctot et al., 2009 as a more specific form of his chance-sampled MCCFR algorithm (MCCFR algorithms are described momentarily).

One problem with CFR is that calculating an agent's expected reward for playing an action requires the entire sub-tree under that action to be traversed, which is computationally costly. A second problem is that for agent actions that lead to opponent actions, CFR needs the opponents' strategies to calculate the agent's expected rewards. It also needs the opponents' strategies to calculate the probabilities that the agent reached its decisions if it had tried to do so. To solve the first problem Lanctot et al., 2009 proposed Monte Carlo Counterfactual Regret Minimisation (MCCFR), a family of sample-based CFR algorithms. MCCFR works by replacing an exact calculation of expected reward with an unbiased estimate. CFR calculates an agent's expected reward for an action as the sum over each reward it could receive after playing it multiplied by the probability of reaching that reward. An MCCFR algorithm performs the same calculation but only for an unbiased sample of the possible rewards. Thus, an agent's expected reward for an action is estimated by traversing only part of the sub-tree under that action, which reduces the computational cost. In expectation MCCFR algorithms perform the same regret updates as the CFR algorithm but require more iterations. However, the cost per iteration is much lower. Lanctot et al., 2009 showed that this generally speeds up convergence and makes the algorithm applicable to larger games. To solve the second problem Lanctot et al., 2009 proposed Outcome Sampling Monte Carlo Counterfactual Regret Minimisation (OS-MCCFR), which is a particular sample-based algorithm that, under some assumptions, does not need to know opponents' strategies, allowing it to be used to minimise regret online.

Counterfactual Regret Minimisation

Zinkevich et al., 2008 proved that, in self-play, the CFR algorithm minimises the maximum counterfactual regret over all information sets and actions in two-player, zero-sum, imperfect information games. To do this, on each iteration, it updates the cumulative counterfactual regret of each action at each information set and uses regret matching to update the strategy profile. The key to calculating counterfactual regrets is calculating counterfactual values. The following definitions show how to calculate counterfactual values, counterfactual regrets and how to perform regret matching.

Counterfactual Value Player i's counterfactual value of information set $I \in \mathcal{I}_i$ given strategy profile σ, $v_i(I|\sigma)$, is the sum of player i's expected reward for each node $\overline{u}_i(h)$ in that information set $h \in I$ multiplied by the probability of reaching that node from the root if each of player i's prior action probabilities were set to one, $\Pr(h|\sigma_{-i})$.

$$v_i(I|\sigma) = \sum_{h \in I} \Pr(h|\sigma_{-i}) \overline{u}_i(h), \tag{3.34}$$

$$\text{where } \overline{u}_i(h) = \frac{1}{\Pr(h|\sigma)} \sum_{z \in Z[h]} \Pr(z|\sigma) u_i(z). \tag{3.35}$$

Here $\Pr(h|\sigma)$ is the probability of reaching node h from the root according to strategy profile σ, $\Pr(z|\sigma)$ is the probability of reaching terminal node z from the root according to strategy profile σ, $Z[h]$ is the set of terminal nodes reachable from node h, and $u_i(z)$ is player i's reward at terminal node z. Note that $\Pr(z|\sigma)/\Pr(h|\sigma)$ is the probability of reaching terminal node z from node h according to strategy profile σ.

Counterfactual Regret Player i's counterfactual regret for not playing action $a \in A(I)$ at information set $I \in \mathcal{I}_i$, $r_i(I, a)$, is the difference between its counterfactual values of I when playing a at I vs playing its strategy σ_i at I

$$r_i(I, a) = v_i(I|\sigma_{I \to a}) - v_i(I|\sigma), \tag{3.36}$$

where $\sigma_{I \to a}$ is the same as σ except action a is played at information set I with certainty. This means that $v_i(I|\sigma_{I \to a}) = v_i(I'|\sigma)$ where $I' = (I, a) \in \mathcal{I}_i$ is the information set reached after playing action a in information set I. Positive regret means that the player would have preferred to play action a rather than its strategy, zero regret means that the player is indifferent between action a and its strategy, and negative regret means that the player preferred its strategy rather than playing action a.

Regret Matching Regret matching is used to update each action probability at an information set by normalising the positive part of its accumulated counterfactual regret as follows

$$\sigma_i^{T+1}(I,a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a' \in A(I)} R_i^{T,+}(I,a')} & \text{if denominator } > 0, \\ \frac{1}{|A(I)|} & \text{otherwise,} \end{cases} \tag{3.37}$$

$$\text{where } R_i^{T,+}(I,a) = \max(R_i^T(I,a), 0), \tag{3.38}$$

$$R_i^T(I,a) = \sum_{t=1}^{T} r_i^t(I,a), \tag{3.39}$$

$\sigma_i^{T+1}(I,a)$ is player i's new probability of playing action a at information set I at iteration $T+1$, $r_i^t(I,a)$ is player i's counterfactual regret of not playing action a at information set I at iteration t, $R_i^T(I,a)$ is player i's cumulative counterfactual regret of not playing action a at information set I between iterations $t=1$ and $t=T$ and $R_i^{T,+}(I,a)$ is the maximum of $R_i^T(I,a)$ and 0. If player i uses the CFR algorithm, then one iteration would calculate the counterfactual regrets for all its actions at all its information sets, use these to update cumulative counterfactual regrets, which in turn would be used with regret matching to update its action probabilities.

Monte Carlo Counterfactual Regret Minimisation

There are two main problems with the CFR algorithm. Firstly, if an agent uses the CFR algorithm, then on each iteration it must calculate the expected reward of each of its actions at each node where it acts, which is computationally costly. Secondly, the other agents' strategies must be known to calculate the expected rewards of the agent's actions that are followed by the other agents' actions, and to calculate the probability of reaching its information sets if the other agents' actions lead to them. The MCCFR family of algorithms are the same as the CFR algorithm except they replace exact expected rewards with unbiased estimates. The exact expected reward for an action is the sum of each reward that could be received after it multiplied by the probability of receiving it. To get an unbiased estimate of this exact expected reward, MCCFR performs the same calculation but only using an unbiased sample of the rewards that could be received after the action. On each iteration, MCCFR only samples parts of the game tree and applies the regret minimisation process to those sampled sub-trees using the estimated counterfactual values instead of the true counterfactual values. Since the estimates are unbiased, the updates involving them approximate the true updates in expectation. The counterfactual regret and regret matching equations remain the same, only the counterfactual value equation changes. Moreover, OS-MCCFR is a particular MCCFR algorithm that also solves the second problem as it does not require the other agent's strategies.

An MCCFR algorithm requires two components to be defined. The first component is a set of subsets of terminal nodes $\mathcal{Q} = \{Q_1, Q_2, \dots, Q_{|\mathcal{Q}|}\}$, such that their union spans the set of terminal nodes $\bigcup_{Q_j \in \mathcal{Q}} Q_j = Z$. The second component is a probability distribution over the subsets in $\mathcal{Q}$, let q_j be the probability of sampling subset Q_j. Also, let $q(z)$ be the probability of sampling terminal node z, it can be calculated by summing the probabilities of the subsets that contain z, $q(z) = \sum_{j:z\in Q_j} q_j$. On each iteration the agent samples one subset Q_j with probability q_j and the information sets containing nodes that are ancestors of the terminal nodes in Q_j are updated. The following definitions show how, in general, to calculate a sampled counterfactual value and how OS-MCCFR works.

Sampled Counterfactual Value Player i's *sampled* counterfactual value of information set $I \in \mathcal{I}_i$ given strategy profile σ is the same as its counterfactual value but only calculated over a subset of terminal nodes Q_j, $\tilde{v}_i(I|\sigma, Q_j)$,

$$\tilde{v}_i(I|\sigma, Q_j) = \sum_{h\in I} \Pr(h|\sigma_{-i})\tilde{u}_i(h|Q_j) \tag{3.40}$$

$$\text{where } \tilde{u}_i(h|Q_j) = \frac{1}{\Pr(h|\sigma)} \sum_{z\in Q_j \cap Z[h]} \frac{1}{q(z)} \Pr(z|\sigma)u_i(z), \tag{3.41}$$

$$q(z) = \sum_{j:z\in Q_j} q_j, \tag{3.42}$$

$\tilde{u}_i(h|Q_j)$ is the sampled expected reward of node $h \in I$, which is sampled over the terminal nodes that are both in Q_j and $Z[h]$. Recall that $Z[h]$ is the set of terminal nodes that are reachable from information set I. Therefore, the sampled expected value of information set I is only calculated over terminal nodes that are in the subset Q_j and that are reachable from I. If the agent samples a Q_j such that $Q_j \cap Z[h] = \emptyset$, then node h is not an ancestor of any terminal node $z \in Q_j$ and $\tilde{u}_i(h|Q_j) = 0$. If $Q_j \cap Z[h] = \emptyset$ for all $h \in I$ then none of the nodes in information set I are ancestors of the terminal nodes in Q_j and $\tilde{v}_i(I|\sigma, Q_j) = 0$.

Outcome Sampling The OS-MCCFR algorithm defines the set of subsets of terminal nodes $\mathcal{Q}$ such that each subset contains exactly one terminal node i.e. $|Q_j| = 1$ for all $Q_j \in \mathcal{Q}$. This means that, on each iteration, only one terminal node is sampled and the information sets along the path from the root to it are updated. The probability of sampling a terminal node $q(z)$ is then equal to the probability of sampling the subset that contains that terminal node $q(z) = q_j$. A probability distribution, or sampling scheme, is selected such that $q(z) = q_j = \Pr(z|\sigma')$. The sampled counterfactual value is then calculated as

$$\begin{aligned}
\tilde{v}_i(I|\sigma, Q_j) &= \sum_{h \in I} \Pr(h|\sigma_{-i}) \tilde{u}_i(h|Q_j) \\
&= \sum_{h \in I} \Pr(h|\sigma_{-i}) \left(\frac{1}{\Pr(h|\sigma)} \sum_{z \in Q_j \cap Z[h]} \frac{1}{q(z)} \Pr(z|\sigma) u_i(z) \right) \\
&= \frac{\Pr(h|\sigma_{-i}) \Pr(z|\sigma) u_i(z)}{\Pr(h|\sigma) \Pr(z|\sigma')} && (3.43) \\
&= \frac{\Pr(h|\sigma_{-i}) \Pr(z|\sigma_i) \Pr(z|\sigma_{-i}) u_i(z)}{\Pr(h|\sigma_i) \Pr(h|\sigma_{-i}) \Pr(z|\sigma'_i) \Pr(z|\sigma'_{-i})} \\
&= \frac{\Pr(h|\sigma_i) \Pr(z|\sigma_{-i}) u_i(z)}{\Pr(h|\sigma_i) \Pr(z|\sigma'_i) \Pr(z|\sigma'_{-i})} \\
&\approx \frac{\Pr(z|\sigma_i) u_i(z)}{\Pr(h|\sigma_i) \Pr(z|\sigma'_i)}. && (3.44)
\end{aligned}$$

Since Q_j only contains one terminal node (i.e. $|Q_j| = 1$) and the probability of reaching this terminal node, $\Pr(z \in Q_j|\sigma)$, is zero for all nodes in I except one (given that I leads to $I' \in \mathcal{I}_i$ where $z \in I'$), the sums can be dropped, giving Equation 3.43. The probability of reaching a node given the strategy profile, can be factored into the probability of reaching that node given player i's strategy multiplied by the probability of reaching that node given the other players' strategies i.e. $\Pr(h|\sigma) = \Pr(h|\sigma_i) \Pr(h|\sigma_{-i})$ and $\Pr(z|\sigma) = \Pr(z|\sigma_i) \Pr(z|\sigma_{-i})$. Finally, by assuming that the sampling strategy profile for the other players is approximately equal to their actual strategy profile i.e. $\sigma'_{-i} \approx \sigma_{-i}$ we arrive at Equation 3.44. This equation for the sampled counterfactual value only depends on the player's strategy, the player's sampling strategy, and the player's utility function. After a game is played and a terminal node is reached, this sampled counterfactual value can be calculated for each of a player's information sets along the path to this terminal node. Each sampled counterfactual value can then be used in place of the true counterfactual value to calculate a counterfactual regret. This counterfactual regret can then be added to a cumulative counterfactual regret, which can be used in regret matching to update the player's strategy at the corresponding information set. A major difference between CFR and OS-MCCFR is that each iteration of OS-MCCFR corresponds to a game being played. An agent using OS-MCCFR does not need to know the opponent's strategy. Instead, it can sample or play games against an opponent.

3.2.10 Opponent Modelling

Fictitious Play

Fictitious play is an algorithm that was originally proposed by Brown, 1951 to explain Nash equilibrium play. It assumes that its opponent is playing a stationary, possibly mixed, strategy and builds an estimate of the opponent's strategy using a frequentist approach. It then plays a best-response to this strategy i.e. a best-response to the opponent's empirical frequencies of play. If its opponent plays a stationary strategy, then as more games are played, its empirical frequencies more accurately approximate that strategy, and in turn its best-response strategy becomes more accurate. Since it builds a model of its opponent, it is also one of the earliest opponent modelling algorithms. It is often used for iteratively approximating solutions for games, where a solution is defined as a best-response strategy, which could also be part of a Nash equilibrium. It estimates its opponent's strategy by normalising the opponent's action counts as follows

$$\tilde{\sigma}^{t}_{\text{opp}}\left(a_{\text{opp}}\right) \leftarrow \left(1 - \frac{1}{t}\right) \tilde{\sigma}^{t-1}_{\text{opp}}\left(a_{\text{opp}}\right) + \frac{1}{t}[\![a_{\text{opp}} = a^{t}_{\text{opp}}]\!] \tag{3.45}$$

where $[\![\cdot]\!]$ is the Iverson bracket such that $[\![\phi]\!] = 1$ if the predicate ϕ is true, otherwise $[\![\phi]\!] = 0$, $\tilde{\sigma}^{t}_{\text{opp}}$ is the estimated opponent strategy at time t, a_{opp} is one of the opponent's possible actions, a^{t}_{opp} is the opponent's actual action at time t, and $\tilde{\sigma}^{t}_{\text{opp}}\left(a_{\text{opp}}\right)$ returns the estimated probability of the opponent's action a_{opp}. In each iteration, fictitious play would predict that the opponent's strategy, σ^{t}_{opp}, is the same as $\tilde{\sigma}^{t}_{\text{opp}}$ and would play a best-response strategy to it i.e. $\sigma^{t}_{\text{FP}} = BR(\tilde{\sigma}^{t}_{\text{opp}})$ in the hope that this will be close to the best-response strategy to the true opponent strategy $BR(\sigma^t_{\text{opp}})$. Against multiple opponents, each opponent's estimated strategy is used to compute and play a best-response strategy. For Equation 3.45 to be valid $t \geq 1$ i.e. the first update occurs at $t = 1$. Fictitious play's initial action at $t = 0$ is arbitrary and is usually selected according to a uniform distribution over all of its available actions i.e. $\tilde{\sigma}^0_{\text{opp}}(a_{\text{opp}}) = 1/|A_{\text{opp}}|$ where A_{opp} is the opponent's set of actions. Subsequent actions are selected uniformly from the set of best-responses calculated using the opponents' estimated strategies. Fictitious play can have its beliefs initialised to specific values by assuming that a number of opponent actions have already been observed and this can affect its convergence as shown for example by Fudenberg and Kreps, 1993 in their persistent miscoordination example (discussed later).

Fudenberg and Kreps, 1993 showed that for fictitious play in self-play, *strict* Nash equilibria are absorbing states. This means that in an iterated game, if a strict Nash equilibrium is played at some point, then it will also be played at all subsequent points. For a strict Nash equilibrium, the inequalities in Equation 3.6 are strict, and so it is always a pure strategy Nash equilibrium. For a weak Nash equilibrium, the inequalities in Equation 3.6 are equalities, and so it is either a pure or a mixed strategy Nash equilibrium. Thus, if in self-play fictitious play converges to a pure strategy profile, then it must be a Nash equilibrium, and if its empirical distributions of plays converge to some (mixed) strategy profile, then that strategy profile must also be a Nash equilibrium. Finally, the empirical distributions of plays of two fictitious players have been shown to converge to Nash equilibria in self-play in: two-player, zero-sum games [Robinson, 1951], two-player, two-action games [Miyasawa, 1961], games with an interior evolutionary stable strategy [Hofbauer, 1995], potential games [Monderer and Shapley, 1996], and certain classes of supermodular games [Hahn, 1999; Krishna, 1992; Milgrom and Roberts, 1990].

However, for fictitious players in self-play, their empirical distributions of plays do not always converge to a Nash equilibrium. Shapley, 1963 showed that this is true in a generalised general-sum version of rock-paper-scissors, called Shapley's game, even though it is not true in unmodified rock-paper-scissors. In Shapley's game, the unique mixed strategy Nash equilibrium is for each player to play each of its actions with equal probability of 1/3. Shapley, 1963 showed that if the players initially select (R, S) then play will follow the sequence ((R, S), (R, P), (S, P), (S, R), (P, R), (P, S), (R, S), . . .). When the strategy profile changes, the player with the losing strategy changes its strategy to a winning strategy. Due to this, the diagonal strategy profiles (i.e. (R, R), (P, P) and (S, S)) are never played giving each player an expected reward of 1/2 rather than the expected reward at the Nash equilibrium of 1/3. Jordan, 1993 also showed that this is true in a three-player version of matching pennies, called Jordan's game, even though it is not true in the original two-player version of matching pennies. Fudenberg and Kreps, 1993 also showed with their persistent miscoordination example that even if its empirical distribution of plays converges, its expected rewards may differ from the expected rewards of the converged to strategy. Specifically, if fictitious play plays against itself in the choosing sides game with certain initial conditions, the empirical joint distribution on pairs of actions will not equal the product of the two marginal distributions. The empirical joint distribution will show correlated rather than independent play. So if the row and column players have initial stored counts of $(1, \sqrt{2})$ and $(\sqrt{2}, 1)$ and initially select (B, A) then play follows the alternating sequence ((B, A), (A, B), (B, A), . . .). The result is that each player's empirical distribution of plays gives each of its actions equal probability of 1/2, but the players never successfully coordinate, each having an expected reward of zero. This behaviour contradicts the assumption that opponents are playing independent and identically distributed actions. Finally, if there are multiple Nash equilibria and it converges to one of them, then it may not converge to the "best" Nash equilibrium. In particular, it may converge to a Nash equilibrium that is objectively worse than another Nash equilibrium, where one or more players would have been strictly better off.

Sequence Prediction

Sequence prediction methods are used to model opponent strategies that may be based on memories of interaction, or may be changing, or both. Their main advantage is that they can recognise correlations in the opponent's play. A sequence prediction method observes a sequence of symbols, also called a context, x^{t-n+1}, $x^{t-n+2}, ..., x^t$ from some alphabet $x^i \in \Sigma$ where $(t-n+1) \leq i \leq t$, t is time and n is the short-term memory size or lookback, and predicts the next observation x^{t+1}. Probabilistic predictions can also be returned based on previous observations i.e. $\Pr(x^{t+1}|x^t, x^{t-1}, ..., x^{t-n})$. Some of this work is an example of Markov process analysis where probabilistic prediction algorithms are used to model Markov processes. All of the sequence prediction methods used in this thesis are modified to be able to return probabilistic predictions given a hypothesised context i.e. $\Pr\left(x^{t+k}|x^{t+k-1}, x^{t+k-2}, ..., x^{t+k-n}\right)$ where $k \geq 1$ is the amount of lookahead and x^i is a hypothesised symbol if $i > t$. For each method this requires saving its current parameters, observing the hypothesised sequence, returning probabilistic predictions and finally restoring its saved parameters. This ensures unobserved sequences do not update the probability distributions.

Sequence Prediction versus Empirical Probability An opponent's strategy can be modelled using empirical probabilities by estimating each opponent action probability at each state as

$$\Pr(a|s) = \frac{f(a|s)}{\sum_{a' \in M(s)} f(a'|s)} \tag{3.46}$$

where $f(a|s)$ is the empirical frequency of action a in state s, and $M(s)$ is the set of opponent actions in state s. In general, an opponent action probability can be conditioned on any or even no information. This information may not be the same as what the state s is perceived to be. For example, in the iterated prisoner's dilemma, the state could be assumed to be the same in each game. This is true for strategies like always defect and always cooperate. However, this is false for more complex strategies like tit-for-tat and pavlov. This highlights a problem with this model, which is that its accuracy will depend on what is believed the opponent perceives as states. Another problem is that it assumes the true probability distribution is fixed.

Alternatively an opponent's strategy can be modelled using sequence prediction, which assumes that an opponent action probability depends on a sequence of previous observations. These observations could be, for example, states or actions or both and the sequence can, in general, be any element from the powerset of the multiset of previous observations. A sequence prediction method still calculates empirical probabilities, but they will be conditioned on this sequence. This somewhat alleviates the first problem in that sequence prediction methods may discover the opponent's perception of state. Some sequence prediction methods tackle the second problem of a changing opponent strategy by discarding probability distributions that no longer adequately make predictions.

Sequence Prediction Model One way to model an opponent's strategy is to use a Markov model. A Markov model is a stochastic model that assumes the Markov property. This property holds if the probability of the future depends only on the present and not on the past i.e. $\Pr(x^{t+1}|x^t, x^{t-1}, \ldots, x^1) = \Pr(x^{t+1}|x^t)$. When applied to opponent modelling, the assumption is that the probability of the opponent's next action only depends on the current state of the environment. This can be expressed as $\Pr(a^t_{\text{opp}}|s^t, a^{t-1}, s^{t-1}, a^{t-2}, s^{t-2}, \ldots, a^1, s^1) = \Pr(a^t_{\text{opp}}|s^t)$ where a^t_{opp} is the opponent's action at time t, a^t is an action belonging to either the player or the opponent at time t, and s^t is the environment state at time t. Sequence prediction methods do not use a Markov model to model the opponent's strategy. Instead, they assume that the probability of an opponent's action can depend on histories of states, or actions, or both. What these histories are depends on the sequence prediction method, but in general they can be any subsequence of the full history of states and actions. This can be expressed as $\Pr(a^t_{\text{opp}}|H)$ where $H \subseteq \{s^t, a^{t-1}, s^{t-1}, a^{t-2}, s^{t-2}, \ldots, a^1, s^1\}$.

Sequence prediction methods usually have two components, a short-term memory and a long-term memory. The short-term memory S is a sequence (ordered list) of the previous $k \in \mathbb{N}^+$ observations i.e. $S = (o^t, o^{t-1}, o^{t-2}, \ldots, o^{t-k})$, where o^t is an observation at time t. Each observation is a symbol, such as a state or action, belonging to some alphabet Σ i.e. $o^t \in \Sigma$ for all t. The long-term memory L is a map from sequences of symbols (lengths 0 to k) and symbols to counts $L : (o_1, o_2, \ldots, o_i) \times \Sigma \to \mathbb{N}^0$, where $0 \leq i \leq k$ and o_i is the i-th symbol in the sequence. The long-term memory can be used to generate a set of distributions, each one conditioned on a different history H i.e. $\{\Pr(a^t_{\text{opp}}|H) : H \subseteq \{s^t, a^{t-1}, s^{t-1}, a^{t-2}, s^{t-2}, \ldots, a^1, s^1\}\}$.

A sequence prediction method that observes o^t does the following. Firstly, it generates a set of histories, where each history is a sequence of symbols, using its short-term memory. Secondly, for each history, it creates or updates a distribution that is conditioned on it using the observation. Finally, it adds the observation to the short-term memory. A sequence prediction method makes a prediction by doing the following. Firstly, it generates a set of histories using its short-term memory. Finally, it predicts using the set of distributions that are conditioned on these histories. Specific details for each of these steps will depend on the method.

For example, consider Entropy Learned Pruned Hypothesis Space (ELPH) [Jensen et al., 2005]. ELPH follows the outlined procedures almost exactly with a few additional steps and implementation details. The way it generates its histories in both the observation and prediction routines is to find the powerset of its short-term memory. The powerset of a set $B = \{b_1, b_2, \dots b_n\}$ is defined as the set of all subsets of B including B itself i.e. $\mathcal{P}(B) = \{\{\}, \{b_1\}, \dots, \{b_n\}, \dots, \{b_1, b_2\}, \dots, \{b_1, b_n\}, \dots, \{b_1, b_2, \dots b_n\}\}$. It has an additional step in the observation routine which, after creating/updating distributions, removes any distribution with a normalised Shannon entropy greater than a user-specified threshold. Additionally, it ensures that the short-term memory size does not exceed a user-specified limit by removing the oldest observations if necessary. Finally, after generating its set of histories, it makes a prediction using a distribution that is conditioned on one of those histories and that has the lowest normalised reliable Shannon entropy.

The Shannon entropy of P is defined as

$$H(P) = -\sum_i P(i) \ln P(i). \tag{3.47}$$

The reliable Shannon entropy of P is calculated by altering the underlying counts that P is assumed to be based on. Given $P(i) = \dfrac{c(i)}{\sum_i c(i)}$, where $c(i)$ is the counts of i, a single count is added for an unknown and new category. The reliable Shannon entropy of P is then defined as

$$\begin{aligned} H_{\text{rel}}(P) = - & \left[\left[\sum_i \frac{c_i}{\sum_j c(j) + 1} \ln \frac{c_i}{\sum_j c(j) + 1} \right] \right. \\ & \left. + \frac{1}{\sum_i c(i) + 1} \ln \frac{1}{\sum_i c(i) + 1} \right]. \end{aligned} \tag{3.48}$$

The (reliable) Shannon entropy of P has a minimum value of 0 and a maximum value of $\ln(m)$, where m is the number of categories in P. Thus, this value can be normalised, giving the normalised (reliable) Shannon entropy, as follows

$$\overline{H_{[\text{rel}]}}(P) = \frac{1}{\ln(m)} H_{[\text{rel}]}(P). \tag{3.49}$$

Sequence Prediction Example Imagine playing a game of iterated matching pennies (see Section 3.1.7) against an opponent. You are the player who wants to match sides, and you use a sequence prediction method at each step to predict your opponent's next action so that you can match it. For simplicity, lets assume that the sequence prediction method is a hierarchical n-gram with a memory size of $k = 2$. The long-term memory of a sequence prediction method can map a maximum of $\sum_{i=0}^{k} |\Sigma|^i$ sequences of symbols and $|\Sigma|$ symbols to counts. In this case, $\Sigma = \{H, T\}$ and $k = 2$, which gives a maximum of 7 sequences, and with 2 symbols, 14 counts in the long-term memory. Initially, the short-term and long-term memories will be empty. As more games are played, the short-term memory will grow to, and remain at, a size of 2, whilst the long-term memory will grow to have a maximum of 14 counts, with 1 count incremented each game.

Table 3.9: An example showing a hierarchical n-gram's short-term and long-term memory updates as well as its predictions in iterated matching pennies.

t	a^t_{pla}	a^t_{opp}	S	Pr(H)	Pr(T)
0	-	-	()	$\frac{1}{2}$	$\frac{1}{2}$
1	h	T	(T)	0	1
2	t	H	(T,H)	$\frac{1}{2}$	$\frac{1}{2}$
3	h	H	(H,H)	1	0
4	h	T	(H,T)	1	0
5	t	T	(T,T)	$\frac{1}{2}$	$\frac{1}{2}$
6	h	H	(T,H)	1	0

	$t=0$		$t=1$		$t=2$		$t=3$		$t=4$		$t=5$		$t=6$	
	H	T	H	T	H	T	H	T	H	T	H	T	H	T
()	-	-	-	1	1	1	2	1	2	2	2	3	3	3
(H)	-	-	-	-	-	-	1	-	1	1	1	1	1	1
(T)	-	-	-	-	1	-	1	-	1	-	1	1	2	1
(H,H)	-	-	-	-	-	-	-	-	-	1	-	1	-	1
(H,T)	-	-	-	-	-	-	-	-	-	-	-	1	-	1
(T,H)	-	-	-	-	-	-	1	-	1	-	1	-	1	-
(T,T)	-	-	-	-	-	-	-	-	-	-	-	-	1	-

Table 3.9 shows what a hierarchical n-gram sequence predictor updates its short-term and long-term memories to over 6 time steps. Initially, the short-term and long-term memories are empty. The short-term memory adds each opponent action observation to its end, but removes the first observation if it grows beyond its size limit, which is 2 in this case. The long-term memory increments at least one count at each time-step, and by the end there has been a total of 15 increments. Probabilities at each time step are calculated by finding the longest,

most recent subsequence of the updated short-term memory that has a count for one or more symbols, and forming a distribution over these counts. For example, at time $t = 4$, $S = (H, T)$, and the longest, most recent subsequence with counts is (T) and so $\Pr(H) = \frac{L((T),H)}{L((T),H)+L((T),T)} = 1$ and $\Pr(T) = \frac{L((T),T)}{L((T),H)+L((T),T)} = 0$.

Sequence Prediction Methods Most of the sequence prediction methods used follow the same general procedure mentioned previously in Section 3.2.10. This procedure is outlined in Algorithm 5. The exceptions that do not clearly correspond with Algorithm 5 are Long Short-Term Memory (LSTM) and Knuth-Morris-Pratt (KMP). In the case of the former, LSTM does not have a short-term memory, its only memory is in the weights of its recurrent neural network. It simply passes each observation to its recurrent neural network as a unit vector input and outputs a distribution over the possible observations. In the case of the latter, KMP is closer to this representation. However, its short-term memory is ideally unbounded and never cleared and so it is not really “short-term”. Its long-term memory stores symbols following the longest match to the end of its short-term memory. This means that, at any time, its long-term memory is only storing the conditional distribution for one context.

Instead of reproducing the full algorithm for each sequence prediction method used, an overall description is provided and the reader is referred to the references for them. The main differences between sequence prediction methods are in how they represent, update, and use their memories. They can generally be split into three categories based on how they represent their long-term memories, firstly, those using tries for their long-term memories.

Lempel-Ziv-1978 (LZ78) [Lempel and Ziv, 1978] for each observation, o, it is added to the short-term memory, S, $S \leftarrow (S, o)$. If S is not in the trie, then S is added to the trie, the frequency of each prefix of S (including S itself) is incremented, and S is cleared, $S \leftarrow ()$. By default, the short-term memory length, k, is unbounded, $k = \infty$. Despite this, since S is cleared after it is added to the trie, S tends to remain small. Finally, a prediction is made using a distribution formed by normalising the frequencies of all (S, o'), where o' is any symbol that has been observed to occur after S.

Prediction by Partial Matching version C (PPMC) [Moffat, 1990] for each observation, o, it is added to the short-term memory, S, $S \leftarrow (S, o)$. If S is not in the trie, then S is added to the trie. Following this, and regardless of

Algorithm 5 A General Sequence Prediction Method

```
Require: Short-term memory size $k$, a set of possible symbols (an alphabet) $\Sigma$.
 1: Initialise long-term memory $L$ as a map from sequences of symbols (lengths
    0 to $k$) and symbols to counts
        $L : (o_1, o_2, \ldots, o_i) \times \Sigma \rightarrow \mathbb{N}^0,\ \text{where } 0 \leq i \leq k.$
 2: Initialise short-term memory $S$ as a sequence of recent observations $S \leftarrow ()$.
 3: function OBSERVE(a symbol $o^t$ at time $t$)      ▷ E.g. a state, an action, etc.
 4:     Generate a family of subsets over $S$ called $F$   ▷ Depends on the method.
 5:     for all $S' \in F$ do
 6:         if $(S', o^t) \in L$ then
 7:             $L(S', o^t) \leftarrow L(S', o^t) + 1$,
 8:         else
 9:             $L(S', o^t) \leftarrow 1$.
10:         end if
11:     end for
12:     Update $S$ by adding $o^t$ to the end of it, $S \leftarrow (S, o^t)$.
13:     if $|S| > k$ then
14:         Remove the first (oldest) symbol from $S$, $S \leftarrow (S(2), \ldots, S(k+1))$.
15:     end if
16: end function
17: function PREDICT
18:     Generate a family of subsets over $S$ called $F$.  ▷ Depends on the method.
19:     for all $S' \in F$ do
20:         Create a probability distribution over symbols
                $\Pr(o \in \Sigma | S') \leftarrow \frac{L(S', o)}{\sum_{o' \in \Sigma} L(S', o')}.$
21:     end for
22:     return A prediction using these distributions.         ▷ Depends on the
    method.
23: end function
```

whether S was already in the trie, the frequency of each prefix of S (including S itself) is incremented. The short-term memory length, k, is bounded, such that S acts as a fixed-size FIFO stack. Finally, a prediction is made by blending probabilities from distributions associated with all subsequences of S.

Transition Directed Acyclic Graph (TDAG) [Laird and Saul, 1994] for each observation, o, it is added to the short-term memory, S, $S \leftarrow (S, o)$. Additionally, for each leaf node in the trie, l, a new leaf node representing o is added onto it if the following conditions are met. Firstly, the probability of visiting l must be above a threshold, and secondly, the depth of l must be less than the short-term memory length, k, which is bounded, such that S acts as a fixed-size FIFO stack. When a leaf node is added to the trie, the "out-count" of its parent and the "in-count" of it are incremented. The probability of the symbol this leaf node represents (given the symbols that lead up to it) is then its "in-count" divided by its parent's "out-count". Finally, a prediction is made using the distribution associated with the node representing S, or the node representing the longest subsequence of S. The size of this distribution is bounded.

ActiveLeZi [Gopalratnam and Cook, 2003] for each observation, o, it is added to two short-term memories, S_1, S_2, $S_1 \leftarrow (S_1, o)$, $S_2 \leftarrow (S_2, o)$. If S_1 is not in the trie, then S_1 is added to the trie, and S_1 is cleared, $S_1 \leftarrow ()$. The frequency of each prefix of S_2 (including S_2 itself) is incremented. The short-term memory length of both short-term memories, k, is bounded such that each acts as a fixed-size FIFO stack. Finally, a prediction is made by blending probabilities from distributions associated with all subsequences of S.

Secondly, those using maps for their long-term memories.

N-Gram [Millington, 2006] for each observation, o, it is added to the short-term memory, S, $S \leftarrow (S, o)$. If the short-term memory has N symbols in it, $|S| = N$, then it creates or updates a distribution conditioned on the first $N - 1$ symbols using the Nth symbol. Its short-term memory length, k, is bounded and equal to N, such that it acts as a fixed-size FIFO stack. Finally, a prediction is made using the distribution associated with the updated short-term memory.

Hierarchical N-Gram is a collection of 1 to N-Grams where a prediction is made using the N-Gram with the largest short-term memory containing the most recent observations that has an associated distribution.

Entropy Learned Pruned Hypothesis Space (ELPH) [Jensen et al., 2005] for each observation, o, it is added to the short-term memory, S, $S \leftarrow (S, o)$. It updates the distributions for the "powerset" of S excluding the newly added symbol. This means that for a given S, it creates or updates distributions for the subsequences $\mathcal{P}(S) \leftarrow \{(), (S(1)), \ldots, (S(|S|)), (S(1), S(2)), \ldots, (S(1), S(|S-1|)), \ldots, (S(1), S(2), \ldots, S(|S-1|))\}$ using $S(|S|)$. If the normalised reliable Shannon entropy of the distribution of any subsequence that has been created or updated is above a threshold, then it is discarded. The short-term memory length, k, is bounded, such that it acts as a fixed-size FIFO stack. Finally, a prediction is made by finding the "powerset" of its short-term memory, and using the distribution with the lowest normalised Shannon entropy that is associated with one of these subsequences.

Finally, methods using other long-term memory representations.

Long Short-Term Memory (LSTM) [Gers, Schraudolph, and Schmidhuber, 2002] each new observation, o, passes as a unit vector through a recurrent neural network. The number of inputs and outputs equals the number of possible symbols (e.g. the number of possible actions), which needs to be known beforehand. Hidden units are LSTM blocks. It does not have a memory length, k, or a separate short-term memory and long-term memory. Instead, its only memory is in the weights of its recurrent neural network. The output is a distribution over the alphabet.

Knuth-Morris-Pratt (KMP) [Knoll, 2009] for each observation, o, it is added to the short-term memory, S, $S \leftarrow (S, o)$. The KMP string matching algorithm [Knuth, Morris, and Pratt, 1977] is used to find the longest matches to the most recent input to S. This means that it finds matches to the subsequence $(S(i), S(i+1), \ldots, S(|S|))$ where i is as large as possible whilst still producing at least one match. For each match, it stores the symbol following that match. Its short-term memory length, k, is ideally unbounded, but may need to be bounded if there are many observations such that it acts as a fixed-size FIFO stack. In this case the short-term memory is not really “short”, but has the same structure as a short-term memory. Finally, a prediction is made using the distribution formed over all the symbols following the matches.

3.2.11 Value Estimation by Lookahead

When playing a game, we would like to maximise our total reward. This requires us to either be given, or to determine a utility function that will inform us of any rewards we may encounter. A simple and unbiased way to assign rewards for most situations is to give a reward of 1 for success, -1 for failure, and 0 for anything else. The aim of an agent is to learn a strategy that leads to successful situations. The credit assignment problem is that of deciding how to value states, or actions, or both, when this reward is delayed. For example, in chess a player could be given a reward of 1 for winning, 0 for a stalemate, and -1 for losing. These rewards are unbiased because the player’s only goal is to win the game. However, there could be around 50 actions before this reward is received. Value estimation techniques offer ways to associate actions with future rewards.

Heuristics can be used to reduce the delay in a reward being received by assigning rewards to intermediate non-terminal states. Using chess as an example again, one heuristic could be the number of pieces the player owns minus the number of pieces the opponent owns. This would reward the player as it took more of the opponent’s pieces. However, it would also discourage the player from making sacrifices that could put it in a better position. One main problem with heuristics is that they do not represent the true value function and can prevent the discovery of this function.

The following sections describe two methods to estimate future rewards. Firstly tree search, which is referred to as *explicit* lookahead. Secondly temporal-difference learning, which is referred to as *implicit* lookahead.

Tree Search - Explicit Lookahead

Tree search (or traversal) is the process of systematically visiting each node in a tree data structure. Usually these methods are classified by the order in which nodes are visited. Examples include breadth-first search, depth-first search, iterative deepening, best-first search, $\alpha - \beta$ search, etc. Minimax can be seen as a specialised form of tree search where a player selects a child node such that it minimises the opponent's maximum reward. Tree search is used in Chapter 4 up to a limited depth and is referred to as *explicit* lookahead. This is because it is used to directly find a sequence of states and actions that will give the maximum total reward. Upon finding this sequence, the first action in the sequence can be played. Repeating the search at the next step, rather than following the original sequence in its entirety, mitigates mistakes in the original search but may be unnecessary if the future is completely deterministic.

Temporal-Difference Learning - Implicit Lookahead

Temporal-difference (TD) learning combines Monte Carlo and dynamic programming ideas. Similarly to Monte Carlo methods, TD methods do not require a model of the environment's dynamics as they learn directly from raw experience. Specifically, they do not require the environment's reward and next-state probability distributions. Similarly to dynamic programming, TD methods update their estimates based on other learned estimates and do not wait for a final outcome (bootstrapping). This allows TD methods to be implemented online in a fully incremental fashion. One of the simplest TD methods, TD(0), updates the value of a state as

$$V(s^t) \leftarrow (1-\alpha)V(s^t) + \alpha(r^{t+1} + \gamma V(s^{t+1})) \tag{3.50}$$

where $V(s^t)$ is the estimated value of the state s at time t, $0 \leq \alpha \leq 1$ is a learning rate, r^{t+1} is the immediate reward received for reaching the state s at time $t+1$, and γ is the discount factor. This equation updates the estimated value of a state by interpolating between its old value and a target value, where the target value is the reward received in the next state and the discounted estimated value of the next state.

TD learning is referred to as an *implicit* form of lookahead because of its bootstrapping. Specifically, the discounting in TD(0) and Q-learning, equations 3.50 and 3.24, can be used as implicit lookahead. One way to do this for a normal-form game (see Section 3.1.3) is to define states as opponent actions. These methods would then learn the values of (opponent action, player action) pairs. These values would be partly controlled by the discount factor. A discount factor of $\gamma = 0$ would learn the player's payoff matrix values, and selecting an action with the maximum TD value or Q-Learning value would be the same as selecting an action using the payoff matrix values with an explicit lookahead of 1. A discount factor greater than zero but less than one, $0 < \gamma < 1$, would lookahead deeper, it would estimate the value of an (opponent action, player action) pair as its payoff matrix value added to a number of future rewards with each subsequent reward being weighted exponentially less. Finally, a discount factor of exactly one, $\gamma = 1$, would estimate these values as their payoff matrix values added to undiscounted future rewards. Lookahead increases as γ increases because a higher value for γ tells TD-Learning and Q-Learning to discount future rewards less.

3.3 Chapter Summary

This chapter has explored background and related work that is necessary to understand this thesis. In particular, it has looked at game theory concepts that are required to understand the approaches in this thesis, their opponents, and the environments that they operate within. It has also looked at learning in games in more detail with a focus on the problems that many learning algorithms face. It finished by formally describing the reinforcement learning, no-regret, and opponent modelling algorithms that are used within the approaches, as well as how lookahead is used.

Chapter 4

Opponent Modelling with Sequence Prediction, Reinforcement Learning, and Lookahead

This chapter, which is based on [Mealing and Shapiro, 2013], focusses on modelling opponents whose strategies are based on memories of interaction, or whose strategies are changing, and using these models to learn high-reward strategies in small simultaneous-move games. These strategies are learnt by using sequence prediction methods to model the opponent's strategy and to predict its actions given a context of past information, with (possibly delayed) rewards learnt from a reinforcement learning method, to lookahead using explicit tree search. There are two types of opponents whose strategies depend on memories of interaction: firstly, those which have a memory of their own actions, and secondly, those which have a memory of both their own and the agent's actions. These opponents arguably better resemble human-like strategies because, rather than sampling from a probability distribution (a task most humans would probably find difficult), they act based on a short-term memory. The opponents with changing strategies are a selection of popular and state-of-the-art reinforcement learning algorithms.

Reinforcement learning algorithms, such as Q-learning by Watkins, 1989, do not explicitly model the opponent's strategy separately from the environment. This can result in an agent's rewards appearing to be random or noisy, when in fact they are deterministic, and can be completely determined by the agent's strategy as well as the opponent's strategy. Also, a limitation of some opponent modelling algorithms, such as fictitious play, is that they do not consider the possibility that the opponent's strategy is based on a sequential memory. It is empirically shown in this chapter that an agent can use its own memory of interaction, i.e. learnt opponent model and learnt action values, with lookahead to play optimally against these opponents. The approach has three parts:

1. A sequence prediction method to model an opponent whose strategy is based on a memory of interaction, or whose strategy is changing. This method predicts the opponent's actions given a (possibly empty) sequence of hypothesised actions.

2. A reinforcement learning algorithm (Q-learning [Watkins, 1989]), to learn the (possibly delayed) values of the agent's own actions for each game state.

3. Exhaustive explicit lookahead with the learnt opponent model and action values to greedily select future action sequences from all possible paths (up to a limited depth) that maximise total expected reward.

Although each individual part has been used before, this combination of parts is unique to this approach. One main difference between this approach and the standard alternative of reinforcement learning in isolation is that this approach builds an explicit opponent model by observing actions rather than just an implicit opponent model by observing rewards. The other main difference is that this approach uses explicit lookahead by tree search rather than just implicit lookahead by expected future reward discounting. This approach is compared against a variety of popular and state-of-the-art reinforcement learning algorithms. Each of these comparison algorithms observe game states such that each game state is defined to include a memory of previous actions taken by either the player or the opponent. Instead of using explicit lookahead, these algorithms use discounting as an implicit form of lookahead (see Section 3.2.11). The empirical results show that this approach generally gains higher average rewards at faster rates in comparison to the reinforcement learning algorithms.

4.1 Sequence Prediction Opponent Modelling

This section summarises sequence prediction opponent modelling, which is described in detail in Section 3.2.10. Sequence prediction opponent modelling uses sequence prediction methods to predict opponent actions. These methods predict the next symbol, or a probability distribution over the next symbol, given a sequence of previous symbols. Each symbol can be set as, for example, a state, or an action. These methods generally form probability distributions conditioned on different subsequences of the given sequence and make predictions using them. The maximum number of possible subsequences is the powerset of the sequence. The size of the powerset scales exponentially with the length of the sequence. Therefore, these methods usually restrict the number of subsequences that they use as well as the size of the sequence.

4.1.1 Short-Term and Long-Term Memory

The short-term memory, S, of a sequence predictor is the sequence of symbols it has observed in the past k time-steps $S = (S(1), S(2), \ldots, S(k))$, where k is the size of its short-term memory. The long-term memory, L, of a sequence predictor is a map from sequences of symbols (between lengths 0 and k) and symbols to counts $L : (o_1, o_2, \ldots, o_i) \times \Sigma \rightarrow \mathbb{N}^0$, where $0 \leq i \leq k$, o_i is the i-th symbol in the sequence, and Σ is an alphabet or set of possible symbols. These mappings can be used to form conditional probability distributions such that the probability of observing a symbol given a sequence is the count of that symbol given that sequence divided by the sum of the counts of any symbol given that sequence i.e. $\Pr(o|S') = \frac{L(S',o)}{\sum_{o' \in \Sigma} L(S',o')}$.

4.1.2 Sequence Prediction Methods

Generally each method updates its long-term memory by getting distributions associated with subsequences of its short-term memory and updating them with an observation. Its short-term memory is then updated to include the observation, and distributions associated with subsequences of the new short-term memory are used to make predictions. The selected sequence prediction methods include:

- **Lempel-Ziv-1978 (LZ78)** [Lempel and Ziv, 1978] and **Knuth-Morris-Pratt (KMP)** [Knoll, 2009], which have unbounded context lengths.
- **Prediction by Partial Matching version C (PPMC)** [Moffat, 1990] and **ActiveLeZi** [Gopalratnam and Cook, 2003], which blend predictions from different context lengths.

- **Transition Directed Acyclic Graph (TDAG)** [Laird and Saul, 1994] and **Entropy Learned Pruned Hypothesis Space (ELPH)** [Jensen et al., 2005], which remove unlikely contexts.
- **N-Gram** [Millington, 2006] and **Hierarchical N-Gram (H. N-Gram)** [Millington, 2006], where the latter is a group of 1 to N-Grams that predicts using the longest context.
- **Long Short-Term Memory (LSTM)** [Gers, Schraudolph, and Schmidhuber, 2002], which is a recurrent neural network and implicitly incorporates an unbounded context length, prediction blending, and context pruning.

For a more detailed description of sequence prediction as well as these sequence prediction methods see Section 3.2.10. More information on these sequence prediction methods can also be found in the references for them.

4.2 Lookahead

The reinforcement learning agents that are used as comparisons to this approach use implicit lookahead. This means that they estimate action values using a form of TD learning and bootstrapping, specifically Q-Learning. The approach in this chapter uses explicit lookahead with opponent action predictions from sequence prediction methods and estimates of the agent's own action values from Q-Learning to directly find the future path(s) with the highest expected total reward. Specifically, an exhaustive breadth-first search is used, which means that the amount of lookahead must be limited because search time grows exponentially with it. Although lookahead can be costly in terms of time and computation, it is sometimes necessary. To show this, consider the iterated prisoner's dilemma. If we did not lookahead by more than one game, then we would always defect since it gives us the highest reward irrespective of the opponent's strategy. However, if we lookahead two games, then we will see that defecting twice only gives us the highest total reward if the opponent cooperates twice, despite the initial defection. This is extremely unlikely to happen against a rational opponent. If we defect against a rational opponent when it tried to cooperate, then it is unlikely to try to cooperate with us a second time unless it is very forgiving. If instead we assume a more realistic opponent strategy that copies the player's previous move (i.e. the tit-for-tat strategy), then we could increase our total payoff by cooperating on the first move and defecting on the second. In fact, using the payoffs in Table 3.7, a lookahead of one would lead us to defect at each time-step giving t total reward, whereas a lookahead of two would lead us to cooperate at each time-step giving $3t$ total reward.

4.3 Games in the Experiments

Three two-player, imperfect information, simultaneous-move games are used in the experiments. The first two are normal-form games, namely rock-paper-scissors and the prisoner's dilemma, with the former being zero-sum and the latter being general-sum. The third is Littman's soccer game, which is a stochastic game. These games are described in Section 3.1.7. It is assumed that each player in these games has the option to use perfect recall, which means that it would never forget revealed information.

4.4 Opponents in the Experiments

Firstly, the opponents used in the iterated rock-paper-scissors experiments play deterministic action sequences repeatedly, and can be seen as variable-order Markov models. For example, an opponent labelled $\{R, P, S\}$ would play rock followed by paper followed by scissors in a loop such that its sequence of actions over games would be $(R, P, S, R, P, S, \dots)$. Secondly, the opponents used in the iterated prisoner's dilemma tournaments are described in Table 4.1. Each opponent can be represented as a finite automaton. Finally, the opponents used in the Littman's soccer game experiments are the popular and state-of-the-art reinforcement learning algorithms explained in Section 3.2.8 including ϵ-greedy Q-Learning, WoLF-PHC, WPL and PGA-APP.

Table 4.1: Iterated prisoner's dilemma opponents.

Name	States	Strategy Description
always-cooperate	1	Always plays cooperate.
always-defect	1	Always plays defect.
anti-tit-for-tat	2	Cooperates on the first move, then plays the opposite of the opponent's previous move.
brainless-alteration	2	Repeats (cooperate, defect).
evil-brainless-alteration	2	Repeats (defect, cooperate).
evil-tit-for-tat	2	Defects on the first move, then copies the opponent's previous move.
general-cooperator	2	Cooperates on the first move, and defects only if the opponent defects after it previously cooperated.
grudger	2	Cooperates if the opponent cooperates, but defects forever if the opponent defects.
pavlov	2	Cooperates on the first move, and switches its action whenever the opponent defects.
tit-for-tat	2	Cooperates on the first move, then copies the opponent's previous move.
evil-two-to-betray	3	Defects on the first move, and defects until the opponent cooperates, at which point it cooperates until the opponent defects twice in a row.
evil-two-to-trust	3	Defects on the first move, and defects until the opponent cooperates twice in a row, at which point it will cooperate until the opponent defects.
two-to-betray	3	Cooperates on the first move, and cooperates until the opponent defects twice in a row, at which point it will defect until the opponent cooperates.
two-to-trust	3	Cooperates on the first move, and cooperates until the opponent defects, at which point it defects until the opponent cooperates twice in a row.
tit-for-two-tats	4	Cooperates on the first move, and cooperates unless the opponent defects twice, at which point it defects unless the opponent cooperates twice.
trust-distrust	4	Keeps a counter starting at 4, which it decrements by 1 to a minimum of 1 for each time the opponent defects, and increments by 1 to a maximum of 4 for each time the opponent cooperates. If the counter is at 4 or 3 it cooperates, otherwise if the counter is at 2 or 1 it defects.
soft-grudger	6	Cooperates if the opponent cooperates, but defects 4 times if the opponent defects followed by 2 cooperates to try to make up.
three-then-punish	6	Cooperates on the first move, and cooperates until the opponent defects 3 times in a row, at which point it defects 3 times in a row.
evil-good-evil-trust	6	Cooperates on the first move then defects and then cooperates. If the opponent cooperates twice from the start, then it will cooperate forever after this sequence, otherwise it will defect forever.
five-is-too-much	7	Cooperates on the first move, then copies the opponent's previous move, except if 5 cooperates occur in a row then it defects.

4.5 The Approach

Algorithm 6 uses lookahead with learnt action values and opponent action predictions to maximise total reward. The scalability of this algorithm depends on the complexities of Q-Learning, the prediction algorithm, and the lookahead technique. In large domains these complexities can be mitigated by abstraction. In the experiments exact exhaustive lookahead in the form of breadth-first tree search dominates and is bounded by $\sum_{t=0}^{T} |A|^t$ where $|A|$ is the maximum number of actions in a state and T is the lookahead depth. For deeper lookahead, other approaches would probably be necessary. Although deeper lookahead may discover more optimal action sequences, it is also more computationally expensive. Additionally, the deeper the lookahead the further into the future predictions are made, which may result in a negative cumulative effect on prediction accuracy if the opponent model is inaccurate. Generally, the space and time complexities of a sequence prediction method scale exponentially with the size of its lookback as it stores and predicts using a number of distributions exponential in its lookback. The set of distributions for a specific lookback value will generally be a subset of the set of distributions for a larger lookback value. Thus, larger lookback values can be beneficial because they consider more distributions and contexts and should be able to accurately model opponents with strategies based on larger memories of interaction. With unlimited computational resources an infinite lookback would be best because it would be able to model an opponent with a strategy based on any memory size, but with limited computational resources the ideal lookback would be just large enough to accurately model the opponent, any larger would be wasteful. In the experiments in sections 4.6.1 and 4.6.2, observations are joint actions $x^t = (a^t_{\text{opp}}, a^t_i)$, and state transitions are deterministic $\Pr(s^t) = 1$. In the experiment in Section 4.6.3, observations are opponent actions $x^t = a^t_{\text{opp}}$, and state transitions are probabilistic $\Pr(s^t) = 0.5$ if players try to move to the same place, otherwise they are deterministic $\Pr(s^t) = 1$.

It is interesting to note that there are some alternate implementations of Algorithm 6 with the same overall idea of using an explicit opponent model with learnt action-values to inform an agent's strategy via lookahead. One option is to use the opponent model when updating an action-value by, for example, weighting the value of the maximum valued action in the next state by the estimated probability that the opponent would reach that state. This would only apply with a discount factor of $\gamma > 0$ because if $\gamma = 0$ then there is no implicit lookahead.

Algorithm 6 Player i using prediction and Q-learning with T step lookahead

```
Require: Q-function $Q : S \times A_i \to \mathbb{R}$, previous state $s^{t-1} \in S$, previous player action $a_i^{t-1} \in A_i$, learning rate $0 \leq \alpha \leq 1$, reward $r \in \mathbb{R}$, discount factor $0 \leq \gamma \leq 1$, current state $s^t \in S$, current prediction algorithm $P_{\text{alg}}^t$ (short-term memory size $n$), current observation $x^t$, lookahead $T$, explore rate $\epsilon$.
1: Update the action-value
       $$Q^t\left(s^{t-1}, a_i^{t-1}\right) \leftarrow (1-\alpha) Q^{t-1}\left(s^{t-1}, a_i^{t-1}\right) + \alpha \left[r + \gamma \max_{a_i^t} Q^{t-1}\left(s^t, a_i^t\right)\right].$$
2: Observe $x^t$ with the prediction algorithm $P_{\text{alg}}^t$.
3: Explore with probability $\epsilon$ such that $a_i^{t+1}$ is set to a random action, otherwise, with probability $(1-\epsilon)$, find the action sequence with the maximum total reward $z^*$ and return $a_i^{t+1}$ from $z^*$ where
       $$z^* \leftarrow \arg\max_{a_i^{t+1:t+T}} \sum_{j \leftarrow 1}^{T} Q^t\left(s^{t+j}, a_i^{t+j}\right) \prod_{j' \leftarrow 1}^{j} \Pr\left(s^{t+j'}\right) \underbrace{\Pr\left(a_{\text{opp}}^{t+j'} | x^{(t+j'-1):(t+j'-n)}\right)}_{\text{predicted by } P_{\text{alg}}^t},$$
   and $y^{i:j} = (y^i, y^{i+1}, ..., y^{i+j})$ assuming $i < j$.
```

However, the current update already implicitly accounts for the opponent's action probabilities as they influence what rewards are backed up. Another option is to use rewards directly in the lookahead instead of the action-values except at the last step. This is because currently, given a discount factor of $\gamma > 0$, each action-value represents the estimated expected future discounted reward of taking that action and so is implicitly looking ahead until the end of the game. Thus, summing these weighted action-values for sequential actions is essentially counting estimated expected future discounted rewards multiple times. It is unknown if either of these options would significantly affect performance, making them possibilities for future research.

4.6 Comparison to Reinforcement Learning Agents

In each experiment the approach using Algorithm 6 is compared against several reinforcement learning agents. In the approach, the methods that are used to model the opponent include:

- Fictitious Play (FP) [Brown, 1951]

- Lempel-Ziv-1978 (LZ78) [Lempel and Ziv, 1978]
- Prediction by Partial Matching version C (PPMC) [Moffat, 1990]
- Transition Directed Acyclic Graph (TDAG) [Laird and Saul, 1994]
- ActiveLeZi [Gopalratnam and Cook, 2003]
- N-Gram [Millington, 2006]
- Hierarchical N-Gram (H. N-Gram) [Millington, 2006]
- Entropy Learned Pruned Hypothesis Space (ELPH) [Jensen et al., 2005]
- Long Short-Term Memory (LSTM) [Gers, Schraudolph, and Schmidhuber, 2002]
- Knuth-Morris-Pratt (KMP) [Knoll, 2009]

The reinforcement learning algorithms that are used as comparisons include:

- ϵ-greedy Q-Learning [Watkins, 1989],
- Win or Learn Fast Policy Hill Climbing (WoLF-PHC) [Bowling and Veloso, 2002],
- Weighted Policy Learner (WPL) [Abdallah and Lesser, 2008] and
- Policy Gradient Ascent with Approximate Policy Prediction (PGA-APP) [Zhang and Lesser, 2010]

Only the first agent is labelled as ϵ-greedy to distinguish it from the Q-Learning algorithm, which is not an agent in itself. In fact, each reinforcement learning agent uses an ϵ-greedy strategy. This selects a random action with probability ϵ, and an action according to the strategy with probability $(1 - \epsilon)$.

Table 4.2 shows the parameters for all agents and experiments. Almost all of the parameters that are shared by the approach and the reinforcement learning algorithms are set to be the same to ensure a fair comparison. The only exception is the discount factor. The discount factor is set differently in the iterated rock-paper-scissors games and the iterated prisoner's dilemma tournaments. The iterated rock-paper-scissors experiment investigates the necessity of a memory against opponents with memory-based strategies. Thus, agents only need to lookahead to the next game. This is why for the approach its discount factor is set to 0, because it does not need to implicitly lookahead as it already has its explicit lookahead set to 1. However, for the reinforcement learning algorithms they have no explicit lookahead and so their discount factors are set to 0.99 to ensure at least an implicit lookahead of 1. In the iterated prisoner's dilemma tournaments the approach and the reinforcement learning algorithms try discount factors set to 0 and 0.99 to test the affects on performance of no implicit lookahead and full implicit lookahead. Another reason 0.99 is used in these two experiments is because higher discount factors were found to give higher rewards in these two games. However, the discount factor for all agents in Littman's soccer game is set to 0.9 to match the value used in Littman, 1994.

Since the opponents in the iterated rock-paper-scissors and iterated prisoner's dilemma experiments are stationary, the training exploration rates for these games are initially set to 1 for a sufficient number of games to explore the models and then are slowly decayed to 0. Whereas since the agents in Littman's soccer game are changing their strategies, the training exploration rate is set to the same constant value used by Littman, 1994. The testing exploration rates are all set to 0 to reduce noise in the comparisons. Given the relative simplicity of the opponents in the iterated rock-paper-scissors and iterated prisoner's dilemma experiments only a small number of training and testing steps are used. Whereas the number of training steps in Littman's soccer game is increased to account for the increased complexity. The number of runs are chosen in all cases to ensure statistical significance. The learning rates are set high to 0.99 in the iterated rock-paper-scissors and iterated prisoner's dilemma experiments as the opponents are stationary. Whereas the learning rate is decayed in Littman's soccer game like in Littman, 1994 as the opponent's are changing their strategies. The approach uses a variety of memory sizes in the iterated rock-paper-scissors and iterated prisoner's dilemma experiments to test their effects on performance. Whereas only a memory size of 1 is used in Littman's soccer game to make it more tractable. The approach uses an explicit lookahead of 1 in iterated rock-paper-scissors as only its memory is being investigated. It also uses an explicit lookahead of 1 in Littman's soccer game to make it more tractable. However, it tries explicit lookaheads of 1 and 2 in iterated prisoner's dilemma tournaments to test the effects on performance. The step-sizes are set relatively high to encourage fast learning, but not too high as to always jump outside the strategy space and are decayed in Littman's soccer game in the same manner as the learning rate in Littman, 1994 to encourage convergence. The losing step-size is always set to double the winning step-size as in Bowling and Veloso, 2002. Finally, the prediction length is always set to 1 as in Zhang and Lesser, 2010.

In experiments 4.6.1 and 4.6.2, memory size n means remembering the previous n opponent and n player actions i.e. $(a_{\text{opp}}^{t-n}, a_{\text{pla}}^{t-n}, a_{\text{opp}}^{t-n+1}, a_{\text{pla}}^{t-n+1}, \ldots, a_{\text{opp}}^{t-1}, a_{\text{pla}}^{t-1})$. In the experiment in Section 4.6.3, it means remembering the previous n opponent actions i.e. $(a_{\text{opp}}^{t-n}, a_{\text{opp}}^{t-n+1}, \ldots, a_{\text{opp}}^{t-1})$. Note that in all tables, lighter gradients indicate better values, N/A refers to an unbounded memory length, and SEM means standard error of the mean.

Table 4.2: Parameters for all agents and experiments. For Littman's soccer game the parameters are based on his [Littman, 1994]. Lookahead and short-term memory size are reduced in the soccer game to make it more tractable. WoLF-PHC step-sizes and PGA-APP prediction lengths are based on their defining papers i.e. Bowling and Veloso, 2002 and Zhang and Lesser, 2010 respectively.

	Iter. rock-paper-scissors	Iter. prisoner's dilemma	Littman's soccer
Prediction and reinforcement learning algorithms			
Training exploration rate	$\min\left(1, \frac{1}{3\times10^{-2}t}\right)$	$\min\left(1, \frac{1}{3\times10^{-2}t}\right)$	0.20
Testing exploration rate	0	0	0
Training steps	1×10^4	1×10^4	1×10^5
Testing steps	1×10^3	1×10^3	1×10^5
Runs	100	100	500
Learning rate	0.99	0.99	$\left(10^{\frac{\log 0.01}{T}}\right)^t$
Prediction algorithms			
Memory size	1-3	1-4	1
Lookahead	1	1, 2, 2	1
Discount factor	0	0, 0, 0.99	0.9
Reinforcement learning algorithms			
Discount factor	0.99	0, 0.99, 0.99	0.9
WoLF-PHC, WPL and PGA-APP			
Step-size	0.05	0.05	$\left(10^{\frac{\log 0.01}{T}}\right)^t$
WoLF-PHC			
Winning step-size	0.05	0.05	$\left(10^{\frac{\log 0.01}{T}}\right)^t$
Losing step-size	0.10	0.10	$2\left(10^{\frac{\log 0.01}{T}}\right)^t$
PGA-APP			
Prediction length	1	1	1

4.6.1 Iterated Rock-Paper-Scissors

This experiment looks to answer the question: is a memory needed to learn a best-response strategy against an opponent whose strategy depends on its previous actions? It also looks to compare the performances of Algorithm 6 and the reinforcement learning algorithms against these opponents. Algorithm 6 and the reinforcement learning algorithms were played against opponents who played deterministic action sequences repeatedly (e.g. R,P,S,R,P,S,...) in iterated rock-paper-scissors games. Additionally, Algorithm 6 was tested using each sequence prediction method, as well as fictitious play, in turn as its prediction algorithm. Table 4.3 shows that each agent cannot learn to play a best-response strategy (converge to an average payoff per step of 1 i.e. win every game) if its memory is less than the opponent's model order (right of the dividing line). This is because it cannot completely learn the opponent's strategy if its memory is less than its opponent's model order. Thus, although it might be able to accurately predict the opponent's actions with a smaller model, in general this will not be possible. Whereas, if its memory is at least as great as the opponent's model order (left of the dividing line), it can learn to play a best-response strategy. Table 4.3 also shows that for all memory sizes and model orders (except memory 1 order 2 and memory 2 order 3) Algorithm 6 with sequence prediction methods, gains the highest payoffs at generally the fastest rates (i.e. has the lowest average times to converge to payoffs). LSTM performs poorly, but has many parameters (e.g. initial weights, activation functions, etc.), which if tuned carefully might improve its performance.

Table 4.3: Shows the performance of Algorithm 6, using each SP-OM algorithm and fictitious play in turn as its prediction algorithm, and the reinforcement learning algorithms against opponents playing deterministic action sequences repeatedly. Agent memory size is shown vertically increasing downward. The opponent's action sequence is shown horizontally with model order increasing rightward. Avg Payoff is the average payoff (equal to the number of games won in this case) per step over 1×10^3 testing steps (± SEM). Avg Time is the average number of training steps required to reach the Avg Payoff (± SEM). The dark line divides players with (in)sufficient memory to model the opponent to the (right) left. Maximum allowed time to reach an Avg Payoff was 210 training steps so times around this value may be larger. Lighter gradients indicate better values i.e. higher payoffs and lower times.

	{R,P,S} Order 1			{R,R,P,P,S,S} Order 2			{R,R,R,P,P,P,S,S,S} Order 3		
	Name	Avg Payoff	Avg Time	Name	Avg Payoff	Avg Time	Name	Avg Payoff	Avg Time
Memory Size 1	ELPH	1 ± 0	14.7 ± 0.6	WoLF-PHC	0.645 ± 0.006	89 ± 5	N-Gram	0.667 ± 0	10 ± 0
	H. N-Gram	1 ± 0	15.3 ± 0.7	PGA-APP	0.644 ± 0.008	59 ± 5	TDAG	0.667 ± 0	10 ± 0
	PPMC	1 ± 0	15.6 ± 0.6	ε Q-Learner	0.635 ± 0.008	22 ± 3	H. N-Gram	0.667 ± 0	11 ± 0.3
	TDAG	1 ± 0	15.9 ± 0.6	PPMC	0.627 ± 0.004	89 ± 8	PPMC	0.667 ± 0	17.7 ± 0.6
	N-Gram	1 ± 0	16.8 ± 0.7	N-Gram	0.622 ± 0.003	46 ± 7	ActiveLeZi	0.667 ± 0	19.2 ± 0.6
	ActiveLeZi	1 ± 0	18.9 ± 0.8	TDAG	0.621 ± 0.003	33 ± 5	ELPH	0.666 ± 0.0003	56 ± 4
	WoLF-PHC	1 ± 0	27 ± 2	H. N-Gram	0.618 ± 0.003	46 ± 7	PGA-APP	0.652 ± 0.005	62 ± 4
	PGA-APP	0.973 ± 0.009	24 ± 2	ELPH	0.617 ± 0.002	210 ± 0	WoLF-PHC	0.646 ± 0.004	71 ± 4
	ε Q-Learner	0.97 ± 0.01	29 ± 2	ActiveLeZi	0.613 ± 0.003	53 ± 7	ε Q-Learner	0.582 ± 0.008	48 ± 6
	WPL	0.87 ± 0.01	74 ± 6	WPL	0.374 ± 0.007	143 ± 7	WPL	0.393 ± 0.008	139 ± 7
	LSTM	0.05 ± 0.04	83 ± 7	LSTM	0.001 ± 0.0004	202 ± 4	LSTM	0 ± 0.0001	194 ± 5
Memory Size 2	ELPH	1 ± 0	10 ± 0	N-Gram	1 ± 0	10 ± 0	WoLF-PHC	0.68 ± 0.01	173 ± 6
	PPMC	1 ± 0	15.3 ± 0.6	TDAG	1 ± 0	10 ± 0	TDAG	0.675 ± 0.0009	10.5 ± 0.2
	TDAG	1 ± 0	15.4 ± 0.6	ELPH	1 ± 0	10 ± 0	ActiveLeZi	0.674 ± 0.0009	22 ± 1
	H. N-Gram	1 ± 0	16 ± 0.6	H. N-Gram	1 ± 0	41 ± 1	PPMC	0.673 ± 0.0009	15.2 ± 0.9
	ActiveLeZi	1 ± 0	17.6 ± 0.7	PPMC	1 ± 0	61 ± 1	H. N-Gram	0.673 ± 0.0009	15.8 ± 0.9
	N-Gram	1 ± 0	19 ± 1	ActiveLeZi	1 ± 0	64 ± 1	N-Gram	0.668 ± 0.002	72 ± 4
	WoLF-PHC	0.98 ± 0.008	91 ± 3	ε Q-Learner	0.92 ± 0.01	45 ± 4	ε Q-Learner	0.64 ± 0.01	56 ± 5
	ε Q-Learner	0.97 ± 0.01	28 ± 2	WoLF-PHC	0.91 ± 0.01	147 ± 8	PGA-APP	0.61 ± 0.01	120 ± 7
	PGA-APP	0.92 ± 0.01	52 ± 3	PGA-APP	0.86 ± 0.01	109 ± 6	ELPH	0.6 ± 0.002	58 ± 4
	WPL	0.65 ± 0.02	105 ± 7	WPL	0.54 ± 0.01	71 ± 6	WPL	0.375 ± 0.009	139 ± 7
	LSTM	-0.04 ± 0.03	115 ± 8	LSTM	0.016 ± 0.0003	210 ± 0	LSTM	-0.003 ± 0.003	118 ± 7
Memory Size 3	ELPH	1 ± 0	10 ± 0	TDAG	1 ± 0	10.1 ± 0.1	TDAG	1 ± 0	10.1 ± 0.1
	PPMC	1 ± 0	15.3 ± 0.6	ELPH	1 ± 0	17.2 ± 0.7	ELPH	1 ± 0	16.3 ± 0.7
	TDAG	1 ± 0	15.7 ± 0.7	N-Gram	1 ± 0	19.1 ± 0.9	N-Gram	1 ± 0	47 ± 2
	H. N-Gram	1 ± 0	15.7 ± 0.6	H. N-Gram	1 ± 0	42 ± 1	H. N-Gram	1 ± 0	68 ± 1
	ActiveLeZi	1 ± 0	17.9 ± 0.7	PPMC	1 ± 0	61 ± 1	PPMC	1 ± 0	100 ± 2
	N-Gram	1 ± 0	19 ± 1	ActiveLeZi	1 ± 0	65 ± 1	ActiveLeZi	1 ± 0	119 ± 2
	WoLF-PHC	0.95 ± 0.01	181 ± 6	WoLF-PHC	0.89 ± 0.01	205 ± 3	WoLF-PHC	0.85 ± 0.01	210 ± 0
	ε Q-Learner	0.94 ± 0.01	37 ± 4	ε Q-Learner	0.87 ± 0.01	71 ± 5	ε Q-Learner	0.84 ± 0.01	84 ± 6
	PGA-APP	0.9 ± 0.02	144 ± 6	PGA-APP	0.87 ± 0.01	179 ± 6	PGA-APP	0.77 ± 0.01	198 ± 3
	WPL	0.63 ± 0.01	98 ± 6	WPL	0.69 ± 0.01	208 ± 2	WPL	0.76 ± 0.01	210 ± 0
	LSTM	-0.106 ± 0.001	72 ± 6	LSTM	0.006 ± 0.0002	205 ± 2	LSTM	0.012 ± 0.001	188 ± 5
N/A	KMP	1 ± 0	13.5 ± 0.5	KMP	1 ± 0	10 ± 0	KMP	1 ± 0	10 ± 0
	LZ78	0.986 ± 0.0004	84 ± 4	LZ78	0.98 ± 0.001	209.2 ± 0.7	LZ78	0.969 ± 0.002	210 ± 0
	FP	-0.335 ± 0.002	60 ± 7	FP	-0.167 ± 0.001	85 ± 6	FP	-0.11 ± 0.0008	92 ± 6

4.6.2 Iterated Prisoner's Dilemma

This experiment looks to answer the question: is lookahead necessary to obtain high total payoffs against an opponent whose strategy depends on its own and the agent's previous actions? It also looks to compare the performances of Algorithm 6 and the reinforcement learning algorithms against these opponents. Algorithm 6 and the reinforcement learning algorithms were played against 20 finite automata (taken from Piccolo and Squillero, 2011 and described in Section 4.4) in separate iterated prisoner's dilemma tournaments. Additionally, Algorithm 6 was tested using each sequence prediction method, as well as fictitious play, in turn as its prediction algorithm. Each tournament's players all faced each other without self-play. Table 4.4a shows that reinforcement learning algorithms with a discount factor (implicit lookahead) of 0, and Algorithm 6 with (explicit) lookahead of 1, have average payoffs per step over all memories of 1.93 and 1.93. Table 4.4b shows that increasing the discount factor to 0.99 and the lookahead to 2, increases these values to 2.32 and 2.62 respectively. This is because with more lookahead agents can predict further ahead into the future, allowing them to, for example, account for opponent reactions to their actions. Table 4.4c shows that giving Algorithm 6 both a lookahead of 2 and a discount factor of 0.99, increases its value further to 2.67, but with an increased average time over all memories of 96 steps compared to 31 steps with just a lookahead of 2. The tables show that using Algorithm 6 with lookahead 2 and sequence prediction methods with sufficient memory gains the highest payoffs at generally the fastest rates. For example, in Table 4.4b the highest payoff of 2.873 (1st place) after 60 steps is for Lempel-Ziv-1978 (LZ78), whereas the highest payoff for a reinforcement learning algorithm of 2.74 (1st place) after 180 steps is for ϵ-greedy Q-Learning.

Table 4.4: Shows the performance of Algorithm 6, using each SP-OM algorithm and fictitious play in turn as its prediction algorithm, as well as the reinforcement learning algorithms when played in iterated prisoner's dilemma tournaments against 20 finite automata (taken from Piccolo and Squillero, 2011 and described in Section 4.4). Agent memory size is shown at the top of each part of the table. Avg Payoff is the average payoff per step over 1×10^3 testing steps ($\pm$ SEM). Avg Time is the average number of training steps required to reach the Avg Payoff ($\pm$ SEM). Position is where an agent finished in its tournament. Lighter gradients indicate better values i.e. higher payoffs, lower times, and lower positions.

(a) Reinforcement learning algorithms with discount factor 0 and SP-OM algorithms with lookahead 1.

Memory Size 1					Memory Size 2			
Name	Avg Payoff	Avg Time	Position		Name	Avg Payoff	Avg Time	Position
PGA-APP	2.03 ± 0.01	30 ± 3	13		PGA-APP	2.01 ± 0.01	30 ± 4	14
ε Q-Learner	1.94 ± 0.01	30 ± 4	16		WPL	1.949 ± 0.008	20 ± 1	17
H. N-Gram	1.94 ± 0.01	30 ± 3	16		H. N-Gram	1.933 ± 0.009	20 ± 2	16
LSTM	1.939 ± 0.009	30 ± 3	16		ActiveLeZi	1.926 ± 0.009	20 ± 1	16
PPMC	1.936 ± 0.009	20 ± 1	16		LSTM	1.925 ± 0.009	20 ± 2	16
WPL	1.932 ± 0.007	20 ± 1	17		ELPH	1.922 ± 0.009	20 ± 2	16
TDAG	1.93 ± 0.01	30 ± 2	16		PPMC	1.921 ± 0.009	30 ± 2	16
ELPH	1.93 ± 0.01	30 ± 4	16		N-Gram	1.92 ± 0.009	20 ± 2	16
ActiveLeZi	1.927 ± 0.009	20 ± 1	16		WoLF-PHC	1.92 ± 0.01	30 ± 4	17
N-Gram	1.926 ± 0.009	20 ± 2	16		TDAG	1.902 ± 0.008	20 ± 2	16
WoLF-PHC	1.89 ± 0.01	20 ± 2	18		ε Q-Learner	1.822 ± 0.007	20 ± 2	18
Memory Size 3					Memory Size 4			
PGA-APP	2.02 ± 0.01	30 ± 3	14		PGA-APP	2.009 ± 0.008	20 ± 2	15
WPL	1.958 ± 0.008	20 ± 3	17		WoLF-PHC	1.978 ± 0.009	20 ± 3	16
WoLF-PHC	1.945 ± 0.009	20 ± 3	17		WPL	1.959 ± 0.007	20 ± 1	17
H. N-Gram	1.931 ± 0.009	20 ± 2	16		TDAG	1.94 ± 0.01	30 ± 3	16
N-Gram	1.931 ± 0.009	30 ± 3	16		PPMC	1.932 ± 0.009	30 ± 3	16
ELPH	1.924 ± 0.009	20 ± 2	16		ActiveLeZi	1.927 ± 0.009	20 ± 2	16
TDAG	1.92 ± 0.009	20 ± 2	16		ELPH	1.924 ± 0.009	20 ± 2	16
LSTM	1.92 ± 0.01	30 ± 3	16		N-Gram	1.92 ± 0.01	30 ± 3	16
PPMC	1.918 ± 0.009	20 ± 3	16		H. N-Gram	1.919 ± 0.009	20 ± 2	16
ActiveLeZi	1.917 ± 0.009	20 ± 2	16		LSTM	1.914 ± 0.008	30 ± 2	16
ε Q-Learner	1.773 ± 0.007	20 ± 1	18		ε Q-Learner	1.764 ± 0.007	20 ± 2	18
N/A								
KMP	1.93 ± 0.01	20 ± 2	16					
LZ78	1.927 ± 0.009	20 ± 2	16					
FP	1.922 ± 0.009	30 ± 3	16					

Table 4.4

(b) Reinforcement learning algorithms with discount factor 0.99 and SP-OM algorithms with lookahead 2.

Memory Size 1					Memory Size 2			
Name	Avg Payoff	Avg Time	Position		Name	Avg Payoff	Avg Time	Position
ε Q-Learner	2.68 ± 0.01	180 ± 5	1		ε Q-Learner	2.74 ± 0.01	180 ± 5	1
TDAG	2.607 ± 0.008	20 ± 1	1		N-Gram	2.73 ± 0.01	30 ± 2	1
N-Gram	2.601 ± 0.007	20 ± 1	1		TDAG	2.72 ± 0.01	20 ± 1	1
ELPH	2.597 ± 0.008	30 ± 2	2		H. N-Gram	2.72 ± 0.01	40 ± 3	1
H. N-Gram	2.561 ± 0.009	30 ± 1	1		ActiveLeZi	2.7 ± 0.01	40 ± 3	1
ActiveLeZi	2.56 ± 0.01	30 ± 1	1		PPMC	2.69 ± 0.01	50 ± 4	1
PPMC	2.52 ± 0.01	30 ± 2	2		ELPH	2.51 ± 0.01	30 ± 2	2
LSTM	2.5 ± 0.01	50 ± 4	6		LSTM	2.5 ± 0.01	40 ± 3	6
WPL	2.31 ± 0.01	30 ± 4	12		WPL	2.34 ± 0.01	40 ± 4	12
PGA-APP	2.17 ± 0.02	30 ± 3	13		PGA-APP	2.18 ± 0.02	40 ± 5	13
WoLF-PHC	2.1 ± 0.02	40 ± 5	13		WoLF-PHC	2.14 ± 0.01	30 ± 3	13
Memory Size 3					Memory Size 4			
TDAG	2.74 ± 0.01	30 ± 3	1		TDAG	2.75 ± 0.01	20 ± 3	1
H. N-Gram	2.74 ± 0.01	40 ± 4	1		H. N-Gram	2.74 ± 0.01	50 ± 4	1
N-Gram	2.72 ± 0.01	40 ± 2	1		ActiveLeZi	2.72 ± 0.01	40 ± 3	1
PPMC	2.72 ± 0.01	50 ± 5	1		PPMC	2.72 ± 0.01	50 ± 4	1
ActiveLeZi	2.7 ± 0.01	40 ± 3	1		N-Gram	2.72 ± 0.01	60 ± 3	1
ε Q-Learner	2.65 ± 0.01	170 ± 5	1		ELPH	2.67 ± 0.01	140 ± 7	1
ELPH	2.54 ± 0.01	40 ± 4	2		ε Q-Learner	2.52 ± 0.01	170 ± 5	3
LSTM	2.47 ± 0.01	40 ± 2	7		LSTM	2.47 ± 0.01	40 ± 2	6
WPL	2.32 ± 0.01	30 ± 4	12		WPL	2.32 ± 0.01	30 ± 3	12
PGA-APP	2.18 ± 0.02	40 ± 4	12		PGA-APP	2.14 ± 0.01	30 ± 4	13
WoLF-PHC	2.14 ± 0.02	40 ± 4	13		WoLF-PHC	2.12 ± 0.01	30 ± 3	13
N/A								
LZ78	2.873 ± 0.008	60 ± 4	1					
KMP	2.75 ± 0.01	20 ± 2	1					
FP	1.76 ± 0.006	20 ± 3	18					

Table 4.4

(c) Reinforcement learning algorithms with discount factor 0.99 and SP-OM algorithms with lookahead 2 and discount factor 0.99.

Memory Size 1					Memory Size 2			
Name	Avg Payoff	Avg Time	Position		Name	Avg Payoff	Avg Time	Position
ε Q-Learner	2.68 ± 0.01	180 ± 5	1		TDAG	2.828 ± 0.009	120 ± 6	1
TDAG	2.63 ± 0.01	60 ± 4	1		N-Gram	2.817 ± 0.008	110 ± 5	1
H. N-Gram	2.62 ± 0.01	70 ± 5	1		H. N-Gram	2.817 ± 0.009	110 ± 5	1
N-Gram	2.61 ± 0.01	60 ± 4	2		ELPH	2.817 ± 0.009	120 ± 5	1
ELPH	2.6 ± 0.01	70 ± 5	3		PPMC	2.803 ± 0.008	120 ± 5	1
PPMC	2.6 ± 0.01	70 ± 4	3		ε Q-Learner	2.74 ± 0.01	180 ± 5	1
ActiveLeZi	2.54 ± 0.01	80 ± 7	4		ActiveLeZi	2.47 ± 0.01	50 ± 4	6
LSTM	2.34 ± 0.02	30 ± 3	12		LSTM	2.4 ± 0.01	30 ± 3	11
WPL	2.31 ± 0.01	30 ± 4	12		WPL	2.34 ± 0.01	40 ± 4	12
PGA-APP	2.17 ± 0.02	30 ± 3	13		PGA-APP	2.18 ± 0.02	40 ± 5	13
WoLF-PHC	2.1 ± 0.02	40 ± 5	13		WoLF-PHC	2.14 ± 0.01	30 ± 3	13
Memory Size 3					Memory Size 4			
TDAG	2.847 ± 0.009	130 ± 5	1		PPMC	2.839 ± 0.009	140 ± 5	1
N-Gram	2.831 ± 0.008	140 ± 5	1		TDAG	2.832 ± 0.009	130 ± 5	1
H. N-Gram	2.83 ± 0.009	120 ± 5	1		H. N-Gram	2.828 ± 0.009	120 ± 5	1
PPMC	2.83 ± 0.01	140 ± 5	1		N-Gram	2.826 ± 0.009	160 ± 4	1
ELPH	2.827 ± 0.009	140 ± 6	1		ELPH	2.82 ± 0.01	180 ± 4	1
ε Q-Learner	2.65 ± 0.01	170 ± 5	1		ActiveLeZi	2.59 ± 0.01	90 ± 8	2
ActiveLeZi	2.51 ± 0.01	60 ± 6	3		ε Q-Learner	2.52 ± 0.01	170 ± 5	3
LSTM	2.35 ± 0.02	40 ± 4	12		LSTM	2.36 ± 0.01	30 ± 3	11
WPL	2.32 ± 0.01	30 ± 4	12		WPL	2.32 ± 0.01	30 ± 3	12
PGA-APP	2.18 ± 0.02	40 ± 4	12		PGA-APP	2.14 ± 0.01	30 ± 4	13
WoLF-PHC	2.14 ± 0.02	40 ± 4	13		WoLF-PHC	2.12 ± 0.01	30 ± 3	13
N/A								
KMP	2.834 ± 0.008	130 ± 5	1					
LZ78	2.59 ± 0.01	90 ± 7	3					
FP	2.35 ± 0.01	30 ± 3	11					

4.6.3 Littman's Soccer Game

This experiment aims to directly compare Algorithm 6 to the reinforcement learning algorithms in a larger, stochastic game. Algorithm 6 was played against the reinforcement learning algorithms in Littman's soccer game [Littman, 1994]. Additionally, Algorithm 6 was tested using each sequence prediction method, as well as fictitious play, in turn as its prediction algorithm. For a description of Littman's soccer game see Section 3.1.7. For Algorithm 6, an instance was created for each game state (defined by player positions and ball possession). Table 4.5 shows that Algorithm 6 wins above 50% of the games on average using any prediction algorithm. Despite its simplicity, fictitious play does quite well, but Prediction by Partial Matching version C (PPMC) has the highest performances.

In this experiment all agents learn and change their strategies over time. Another option would be to use agents with stationary strategies. A stationary strategy could be learnt by, for example, playing a reinforcement learning algorithm in self-play for a number of iterations and saving one of the resulting strategies. In general, a learning agent is more likely to learn to win against an opponent with a stationary strategy rather than an opponent with a changing strategy. This is because against the former all opponent action observations/rewards are relevant to build an explicit/implicit opponent model, whereas against the latter only the most recent opponent action observations/rewards are likely to be relevant. With more opponent action observations/rewards its explicit/implicit opponent model and best-response strategy is likely to be more accurate. Thus, testing Algorithm 6 against the reinforcement learning algorithms as they are learning is arguably a more difficult test than if it were to be tested against agents with stationary strategies. In either case it is essential for an agent's strategy to be non-deterministic otherwise, if the opponent learnt its strategy, then the opponent would be able to always block/score.

Table 4.5: Shows the average payoff (fraction of goals scored/games won) per game (Avg Payoff) over 1×10^5 testing games (± SEM) by Algorithm 6, using fictitious play and each SP-OM algorithm in turn as its prediction algorithm vs each reinforcement learning algorithm in Littman's soccer game. Lighter gradients indicate better values i.e. higher payoffs.

ε Q-Learner		WoLF-PHC		WPL		PGA-APP	
Name	Avg Payoff	Name	Avg Payoff	Name	Avg Payoff	Name	Avg Payoff
PPMC	0.687 ± 0.006	PPMC	0.701 ± 0.006	PPMC	0.717 ± 0.004	PPMC	0.648 ± 0.006
LSTM	0.635 ± 0.004	LSTM	0.638 ± 0.005	H. N-Gram	0.674 ± 0.002	H. N-Gram	0.608 ± 0.003
TDAG	0.63 ± 0.004	FP	0.637 ± 0.004	N-Gram	0.665 ± 0.001	ActiveLeZi	0.599 ± 0.004
H. N-Gram	0.628 ± 0.003	N-Gram	0.614 ± 0.003	LSTM	0.659 ± 0.003	FP	0.593 ± 0.003
LZ78	0.621 ± 0.004	H. N-Gram	0.612 ± 0.003	TDAG	0.659 ± 0.002	TDAG	0.589 ± 0.004
N-Gram	0.62 ± 0.003	ActiveLeZi	0.606 ± 0.004	FP	0.655 ± 0.003	LSTM	0.585 ± 0.004
ActiveLeZi	0.618 ± 0.003	TDAG	0.606 ± 0.004	LZ78	0.653 ± 0.002	N-Gram	0.582 ± 0.003
ELPH	0.601 ± 0.004	LZ78	0.602 ± 0.004	ActiveLeZi	0.651 ± 0.002	LZ78	0.574 ± 0.003
FP	0.536 ± 0.003	ELPH	0.576 ± 0.003	ELPH	0.637 ± 0.002	ELPH	0.565 ± 0.003
KMP	0.524 ± 0.002	KMP	0.564 ± 0.003	KMP	0.62 ± 0.002	KMP	0.553 ± 0.003

4.7 Chapter Summary

This chapter has proposed an approach for learning high-reward strategies against opponents whose strategies are based on memories of interaction, or whose strategies are changing in small-simultaneous move game. This approach has three parts. Firstly, adapting and applying a sequence prediction method to model this type of opponent, secondly, learning the agent's own (possibly discounted) rewards using a reinforcement learning method, and finally, using the learnt model and rewards to explicitly lookahead using tree-search. More lookahead allows agents to consider further into the future. Making accurate predictions about this future in this approach is dependent on having accurate opponent models and accurate reward estimates. Empirical results show that given enough memory and lookahead, this approach generally gains higher rewards at faster rates against opponents of variable memory sizes compared to popular and state-of-the-art reinforcement learning algorithms. Additionally, this approach gains higher rewards when played directly against the reinforcement learning algorithms in a large zero-sum soccer game. However, the higher rewards do come at the cost of higher time complexities compared to the reinforcement learning algorithms. In particular, the time and space complexities of a sequence prediction method can grow exponentially with its lookback and the time complexity of exhaustive breadth-first search lookahead grows exponentially with its depth. Thus it is necessary for the lookback and lookahead to be set sufficiently small for the approach to have a time complexity comparable to that of the reinforcement learning algorithms.

The results do not show that any single sequence prediction method is objectively better or worse than the others across all experiments. However, they do show that PPMC has the highest rewards against the reinforcement learning algorithms in Littman's soccer game whereas KMP has the lowest. In the iterated rock-paper-scissors games, many sequence prediction methods are able to completely model the opponents given sufficient memory and thus to play best-response strategies with small variations in the times taken to do this. Overall ELPH, KMP, and TDAG are amongst the best methods in these games whereas FP and LSTM are amongst the worst. In the iterated prisoner's dilemma tournaments all methods perform similarly with a discount factor of 0 and a lookahead of 1. When increasing the lookahead to 2 KMP, LZ78, and TDAG are amongst the best methods whereas FP and LSTM are amongst the worst. When also increasing the discount factor to 0.99 KMP, PPMC, and TDAG are amongst the best methods whereas FP and LSTM are amongst the worst.

Chapter 5

Opponent Modelling with Expectation-Maximisation, Sequence Prediction, and No-Regret Learning

This chapter, which is based on [Mealing and Shapiro, under review], builds on the work in Chapter 4 by using sequence prediction to model changing opponent strategies in larger domains with hidden information. Specifically, this chapter looks at a pair of two-player, imperfect information, zero-sum, turn-based (sequential-move) poker games of medium-size, namely die-roll poker and Rhode Island hold'em. In previous games, including rock-paper-scissors, the iterated prisoner's dilemma, and Littman's soccer game [Littman, 1994], the hidden information is the opponent's actions as both players select their actions simultaneously. Since these poker games are turn-based, there is no uncertainty in the opponent's actions. Instead, the hidden information is in the context, or state, that the opponent takes its actions. This hidden information originates from each player being assigned private information, usually at the beginning but also sometimes during the game. It is assumed that the opponent's hidden information is at least partially revealed at the end of some games. Specifically, either its exact hidden information, or a possible subset of it, is revealed at a showdown, where neither player has folded (given up) during the game. It is also assumed that each player can have perfect recall if it chooses to, meaning it does not forget any information it chooses to observe.

The purpose of an opponent model is to predict the opponent's actions given its information. Thus, learning an opponent model requires observations of the opponent's actions with its corresponding information. However, in this case, the opponent has hidden information, which may only be partially revealed at the end of each game. The actions of a typical opponent will give indications of its hidden information i.e. often betting with strong hands and folding with weak hands. The first proposal is then to infer its hidden information, when it is not revealed, based on its actions using *expectation-maximisation*. This is an iterative procedure to compute maximum likelihood estimates of model parameters given partially observed data. In this case, the model is of the opponent's strategy, the observed data is the opponent's actions given the agent's information (public actions and the agent's hidden information), and the hidden data is the opponent's hidden information. The opponent's strategy is not assumed to be stationary. The second proposal is then to use *sequence prediction* to predict a changing opponent strategy such that for each of its decision points, identified using its inferred hidden information, its actions are predicted using its actions at that point from previous games. Sequence prediction would find effective predictive contexts amongst different interaction memories. Finally, even with an opponent model, you still need to decide how to use it to improve your strategy, which is more difficult if it has inaccuracies. The third proposal is then to *simulate games* against the opponent model. Assuming the agent can learn from games, then this will improve its strategy against the opponent model, which if accurate, will improve its strategy against the opponent. Simulating games is advantageous because it lets you choose the number of simulations to control the computational cost as well as how much the opponent model is relied on. Additionally, the agent's strategy can be updated using any algorithm that uses the rewards from games.

In short, three proposals are put forward in this chapter, which can be used online; two for building an opponent model, specifically to handle hidden information (1) as well as changes in the opponent's strategy (2), and one for using an opponent model, which may have inaccuracies (3). The three proposals are as follows:

1. To use expectation-maximisation to infer the opponent's hidden information when it is not revealed.

2. To use sequence prediction to model the opponent's strategy and predict

its actions based on its inferred hidden information and its actions from previous games.

3. To simulate games against the opponent model in-between games against the opponent to improve learning.

The expectation-maximisation algorithm works as follows. It models the opponent's strategy as a set of categorical distributions, one for each of its decision points over its actions at that decision point. At the end of a game, if the opponent's hidden information is not fully observed, then the decision points that it acted at are unknown. In this case, it treats each opponent action as a sample from a mixture of the categorical distributions associated with the decision points that it could have acted at. It then: 1. infers a distribution over the opponent's hidden information using the categorical distributions parameters (E-step), and 2. updates the categorical distributions parameters by maximising their likelihood given the inferred opponent's hidden information distribution (M-step). Finally, the opponent's hidden information is inferred by sampling from the distribution over it that was inferred in the E-step. Although these categorical distributions could be used to predict the opponent's actions, this would not consider its strategy changing. This is why instead, a sequence prediction method is used, which can account for changes in its strategy.

A sequence prediction method is used as follows. An instance of it is created for each opponent decision point. After each game, the opponent's decision points are worked out using its hidden information, which was either observed or inferred by the expectation-maximisation algorithm. The sequence prediction method instances associated with those decision points observe the opponent's actions taken at them. For each opponent decision point, the associated sequence prediction method instance can predict a distribution over the opponent's actions, a_{opp}, at it conditioned on its knowledge, I, as well as a sequence of previous opponent actions at that decision point, $(a^1_{\text{opp}}, a^2_{\text{opp}}, \dots)$, i.e. $\Pr(a_{\text{opp}} | I, (a^1_{\text{opp}}, a^2_{\text{opp}}, \dots))$.

For every game played against the opponent, a number of games are simulated against the opponent model to try to improve the agent's strategy. In each simulated game, opponent actions are chosen by sampling from opponent action distributions predicted by the sequence prediction method instances, and a state-of-the-art no-regret learning algorithm is used to update the agent's strategy using rewards from actual and simulated games. If the opponent model is completely accurate, then playing a best-response strategy against it would maximise the agent's expected rewards. So why does it not play a best-response strategy against the opponent model each game? The reason is that the opponent model is unlikely to be completely accurate, particularly near the beginning of interaction when it only has data from a few games. This matters because Johanson, Zinkevich, and Bowling, 2008 showed that even a slightly inaccurate best-response strategy can have a very low expected reward. This proposal will exploit the opponent less if the opponent model is completely accurate, but is likely to be less exploitable if it is inaccurate. Many opponent models require knowledge outside the rules of the game, or assume that the opponent's strategy is stationary, or both. This opponent model has several advantages: 1. it can be built and used online; 2. it does not require knowledge outside the rules of the game; 3. it can infer the opponent's hidden information via expectation-maximisation; 4. it can predict the actions of an opponent with a changing strategy via sequence prediction and 5. it can be used with any strategy update method that only requires results from games.

The three proposals are tested in a pair of two-player simplified poker games against various opponents. However, the proposals are applicable to situations with more than two agents by modelling each agent separately, and training against all of them. The primary idea is that the proposals will give higher average payoffs per game than not using them. The secondary ideas are as follows. Firstly, that inferences of the opponent's hidden information based on its behaviour using expectation-maximisation will give higher average payoffs per game in the approach than inferences ignoring its behaviour. Secondly, predictions of the opponent's actions using a sequence prediction method will give higher average payoffs per game in the approach than predictions using empirical probabilities. Experiments in the pair of simplified poker games measuring the changes in the agent's average payoff per game confirm these ideas.

5.1 Opponent Modelling in Poker

A large part of opponent modelling research in games with hidden information, otherwise known as *imperfect information* games, has focused on poker due to its huge popularity. Some approaches use domain-specific heuristics and expert knowledge. For example, Billings et al., 1998 propose a multi-player Texas hold'em agent named Loki, whose strategy is based on poker-specific heuristics, i.e. effective hand strength, which is calculated using hand strength, hand potential, pot odds, and opponent models. Other approaches use large databases of human play. For example, the opponent modelling by Billings et al., 1998 is improved by Davidson et al., 2000 through experiments with neural networks trained on hands played in the Internet Relay Chat (IRC) poker server. A second example is by Ponsen et al., 2008, where they use games played in an online multi-player no-limit Texas hold'em room to learn a relational regression tree-function to adapt prior opponent models to specific opponents. A third example is by Broeck, Driessens, and Ramon, 2009, where they apply Monte Carlo Tree Search (MCTS) to multi-player no-limit Texas hold'em, and learn opponent models using games played in an online casino. A final example is by Rubin and Watson, 2010, where they look at a two-player limit Texas hold'em agent named SARTRE (Similarity Assessment Reasoning for Texas hold'em via Recall of Experience), which acts by re-using solutions similar to its situation from a large database of human poker hands.

Many approaches use Bayesian probabilistic models. For example, Korb, Nicholson, and Jitnah, 1999 propose a Bayesian Poker Program for two-player five-card stud poker, which learns through experience using a Bayesian network to model each player's hand, opponent behaviour conditioned on its hand, and betting curves that govern play given a probability of winning. A second example is by Southey et al., 2005, where they propose a Bayesian probabilistic opponent model for two-player poker games, which infers a posterior opponent strategy given a prior and observations of its play. A final example is by Baker and Cowling, 2007, where they use Bayesian opponent modelling in multi-player one-card poker to classify opponents based on their behaviour as loose or tight, as well as passive or aggressive, and to counter the most dangerous type.

Another set of approaches use best-response strategies, or approximate Nash equilibrium strategies, or both. For example, Risk and Szafron, 2010 use approximate Nash equilibrium strategies in three-player limit Texas hold'em, which they find using counterfactual regret minimisation. Two more examples are by Johanson and Bowling, 2009; Johanson, Zinkevich, and Bowling, 2008, firstly using Restricted Nash Response (RNR) strategies, and secondly using Data Biased Response (DBR) strategies, the latter being an enhancement of the former, which they also find using counterfactual regret minimisation. RNR and DBR strategies tradeoff between exploiting an opponent and being exploitable by solving a modified game to potentially achieve strategies with lower exploitability for a given degree of exploitation. Ponsen, Lanctot, and Jong, 2010 use Monte Carlo sampling to speed up the convergence of RNR strategies. A fourth example is by Bard et al., 2013, where they compute a set of RNR and DBR strategies against certain opponents offline and find the mixture that maximises their expected reward online using a multi-armed bandit algorithm. A final example is by Ganzfried and Sandholm, 2011, where they propose Deviation Based Best-Response, which initialises prior opponent action distributions as if they have played a number of fictitious hands according to an approximate Nash equilibrium strategy, and then updates them through observations of their play. It uses these posterior distributions to compute an opponent model that is close to the approximate Nash equilibrium, making it less exploitable, and plays a best-response strategy against it.

For more information the reader is referred to the review by Sandholm, 2010 on the state of solving incomplete-information games, and the review by Rubin and Watson, 2011 on algorithms, approaches, and agents in computer poker. The expectation-maximisation algorithm in this approach is related to approaches that use Bayesian probabilistic models in that it makes use of Bayes' rule. Additionally, the state-of-the-art no-regret algorithm in this approach is based on counterfactual regret minimisation, which is an algorithm that is also used by[1] to calculate best-response strategies, approximate Nash equilibria, and combinations between both. This work differs from[2] in that it does not use knowledge outside the rules of the game and the opponent model is updated online using only information accessible to the agent. This work also differs from[3] in that it is not assumed that the opponent uses a stationary strategy. One advantage of these differences is that it makes this work applicable to more opponents and more imperfect information turn-based games (or situations that can be modelled as such). Another advantage is that by simulating games against the opponent model, instead of immediately playing a best-response strategy against it, which

[1]Bard et al., 2013; Johanson and Bowling, 2009; Johanson, Zinkevich, and Bowling, 2008; Ponsen, Lanctot, and Jong, 2010; Risk and Szafron, 2010.

[2]Baker and Cowling, 2007; Billings et al., 1998; Broeck, Driessens, and Ramon, 2009; Davidson et al., 2000; Korb, Nicholson, and Jitnah, 1999; Ponsen et al., 2008; Rubin and Watson, 2010; Southey et al., 2005.

[3]Bard et al., 2013; Ganzfried and Sandholm, 2011; Johanson and Bowling, 2009; Johanson, Zinkevich, and Bowling, 2008; Ponsen, Lanctot, and Jong, 2010; Risk and Szafron, 2010.

Johanson, Zinkevich, and Bowling, 2008 showed can be brittle, the strategy will be more robust to inaccuracies in the opponent model. Out of the prior exploitation approaches designed to model dynamic opponents in real-time, only the MCTS approach by Broeck, Driessens, and Ramon, 2009 reports effective results. If their approach did not require prior knowledge in the form of training its opponent model using a large database of games, it could have served as a fair comparison to this approach.

5.2 Expectation-Maximisation

The EM algorithm, first proposed by Dempster, Laird, and Rubin, 1977, can iteratively calculate maximum likelihood estimates of parameters in a statistical model dependent on latent (unobserved) variables. It alternates between an expectation (E) step and a maximisation (M) step. The E-step creates a function for the expectation of the log-likelihood evaluated using current parameter estimates. The M-step updates parameters by maximising the expected log-likelihood computed in the E-step. These new parameters are then used to determine the probability distribution of the latent variables in the next E-step and the algorithm iterates. The EM algorithm will always converge and will always produce a, possibly local, maximum likelihood estimate. This estimate may be improved through multiple runs with different initialisations. The E and M steps are iterated until a termination condition is met (e.g. convergence, a time limit, a computation limit, etc.).

Designing an EM algorithm can be broken down into three steps. The first step is to construct a model for the observed data by writing down an equation for its log-likelihood. This usually includes the log of a sum over the possible values of the hidden data. The sum prevents the log from acting directly on the joint distribution of the observed and hidden data, resulting in complicated maximum likelihood solutions. If the hidden data was known, then the log would act directly on the joint distribution. The second step is to determine the E-step, which replaces the hidden data with its expected value calculated using the current model parameters, which are assumed to be correct. The final step is to determine the M-step, which maximises the likelihood of the model parameters by assuming that the hidden data calculated in the E-step is correct.

An online EM algorithm estimates the hidden data and model parameters on-the-go, without needing to store the observed data, and by continuously updating the estimates with each new observation. The standard EM algorithm requires the entire set of observations to be accessible on each iteration. This makes it impractical when there is a lot of data, or when the data is being streamed, or both. Thus, online EM algorithms bypass this limitation. Liang and Klein, 2009 provide a good overview of online EM algorithms, and this chapter uses what they refer to as a stepwise EM algorithm. The stepwise EM algorithm was developed by Sato and Ishii, 2000 and generalised by Cappé and Moulines, 2008. It stochastically approximates the E-step to add each new observation iteratively and leaves the M-step unchanged. The stepwise EM algorithm of this thesis is inspired by Cappé and Moulines, 2008 as well as by the application of their algorithm by Butterworth, 2010 to poker. The stepwise EM algorithm is guaranteed to converge to a, possibly local, maximum likelihood estimate as long as the step size follows the standard conditions from the stochastic approximation literature. These conditions restrict the step size η_t such that $\sum_{t=0}^{\infty} \eta_t = \infty$ and $\sum_{t=0}^{\infty} \eta_t^2 < \infty$, where t is the parameter update number (i.e. initialisation is $t = 0$ and first update is $t = 1$). For the online EM algorithm proposed in this chapter the step size, η_t, is set to $\eta_t = \frac{1}{t}$ and the sufficient statistics are used to predict the opponent's hidden information.

5.3 Games in the Experiments

A pair of two-player, imperfect information, zero-sum, turn-based poker games are used in the experiments namely die-roll poker and Rhode Island hold'em. It is assumed that the players in both games can have perfect recall if they choose, meaning that they can remember the exact sequence of observable actions. In both of these poker games each player has, at most, three actions when on turn. Each player can either fold (F), giving up the pot, or call (C), matching its opponent's current bet, or raise (R), matching and exceeding its opponent's current bet by a fixed amount. If no one folds, then a showdown eventually occurs and the player with the best hand (composed of dice or cards) wins the pot. When playing Rhode Island hold'em, each player assumes that it is playing a version of the game that has been abstracted using percentile bucketing based on expected hand strength squared. The reason for this is that players one and two each have too many information sets on turn, 2.50×10^7 and 2.46×10^7 respectively, to learn effective strategies within 1×10^5 games, which is the number of games chosen to evaluate agents over in the experiments. Evidence for this is shown in Section 5.6.1. All of these games are fully described in Section 3.1.7.

5.4 Opponents in the Experiments

This chapter uses opponents based on popular and state-of-the-art no-regret and reinforcement learning algorithms in the experiments. These opponents are as follows:

- OS-MCCFR (without an opponent model) by Lanctot et al., 2009.
- PGA-APP (Policy Gradient Ascent with Approximate Policy Prediction), a state-of-the-art, Q-Learning based, reinforcement learning method by Zhang and Lesser, 2010.
- UCB (Upper Confidence Bounds), a popular adaptive bandit algorithm, see Auer, Cesa-Bianchi, and Fischer, 2002.
- CFRX (CFR with X iterations) by Zinkevich et al., 2008, not an agent in itself but used to generate approximate Nash equilibrium strategies (only used in die-roll poker).

Since Upper Confidence Bounds (UCB) is designed for a single-state environment, an instance of it is used for each of the opponent's information sets where it acts. The average reward for each UCB instance is set to the average of the rewards received in the games involving its associated information set. OS-MCCFR is used as an opponent because firstly it is a state-of-the-art no-regret learning algorithm, and secondly because this chapter tests if using it with an opponent model can improve its average payoff per game. PGA-APP is used as an opponent because it is a state-of-the-art reinforcement learning algorithm. UCB is used as an opponent because it is a popular and well-understood adaptive bandit algorithm. Finally, CFRX is used as an opponent because by varying the number of iterations X, a variety of opponents can be created with increasing strengths as X increases. This is due to the theoretical guarantee that the average strategy of CFR will converge to an ϵ-Nash equilibrium strategy in self-play with ϵ decreasing as the number of iterations increases [Zinkevich et al., 2008]. This chapter tests if the opponent model improves the average payoff per game against these opponents. The agent using this approach is labelled OS-MCCFR with an Opponent Model or just OS-MCCFR OM.

5.5 The Approach

5.5.1 Expectation-Maximisation in the Opponent Model

The opponent's strategy σ_{opp}, is a set of discrete probability distributions, one for each of its information sets where it acts $\sigma_{\text{opp}} = \{f_{A(I)} : I \in \mathcal{I}_{\text{opp}} \text{ and } P(I) = \text{opp}\}$, where $f_{A(I)}$ is a probability mass function over $A(I)$. To model it, a set of sequence predictors $\mathcal{P}$ are created, one for each of the opponent's information sets where it acts $\mathcal{P} = \{p_I : I \in \mathcal{I}_{\text{opp}} \text{ and } P(I) = \text{opp}\}$. Each sequence predictor p_I observes the opponent's actions in its associated information set I over games and predicts a discrete probability distribution over the opponent's future actions in that information set. The problem is that in order to know which opponent information sets the opponent acted in, its hidden information needs to be known, which is only sometimes revealed at the end of a game. Thus, this approach waits until the end of a game before updating the opponent model. If the opponent's hidden information is not fully revealed, then it is predicted. The observation or prediction of the opponent's hidden information allows its information sets that it acted in to be identified and the associated sequence predictors to observe the actions taken in them.

The first question is: how is the opponent's hidden information predicted? The opponent's hidden information is predicted by sampling from a probability distribution over its possible instances of hidden information. Recall from Section 3.1.3 that a node or history can be represented as a unique sequence of actions taken to reach it $h = (a_1, a_2, \ldots, a_m)$, where each action a_i, $1 \le i \le m$, is taken by one of the players. An information set can also be represented as a sequence of actions, except some of those actions are hidden. For example, in die-roll poker a node could be $h =$ (⚁, ⚃, r, c, ⚄, ⚂, c) where player one rolled two, player two rolled four, player one raised, player two called, player one rolled five, player two rolled three and player one called. At this point, neither player has seen the other's die-rolls. From player two's perspective, its information set would be (D_1, ⚃, r, c, D_3, ⚂, c) $\in \mathcal{I}_2$ where D_1 and D_3 are player one's hidden six-sided die-rolls. From player one's perspective, its information set would be $(\text{⚀}, D_2, r, c, \text{⚄}, D_4, c) \in \mathcal{I}_1$ where D_2 and D_4 are player two's hidden six-sided dice-rolls.

Let player i's information set $I = \{\mathcal{H}_i, S\} \in \mathcal{I}_i$, where $\mathcal{H}_i$ is its private or hidden information and S is the sequence of actions visible to both players. Using the last example, write $(D_1, \text{⚃}, r, c, D_3, \text{⚁}, c) = \{\mathcal{H}_2, S\} = \{(\text{⚃}, \text{⚁}), (r, c, c)\} \in \mathcal{I}_2$ and $(\text{⚀}, D_2, r, c, \text{⚄}, D_4, c) = \{\mathcal{H}_1, S\} = \{(\text{⚀}, \text{⚄}), (r, c, c)\} \in \mathcal{I}_1$.

Using this notation, the agent observes its own information set $\{\mathcal{H}_{\text{pla}}, S\} \in \mathcal{I}_{\text{pla}}$ and wants to predict the opponent's information set $\{\mathcal{H}_{\text{opp}}, S\} \in \mathcal{I}_{\text{opp}}$. Since the public actions S are already known, only the opponent's hidden information $\mathcal{H}_{\text{opp}}$ needs to be predicted. Using Bayes' rule the agent can predict the probability of the opponent's hidden information given its hidden information and the public actions as

$$\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S) = \frac{\Pr(S|\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})}{\sum_{\mathcal{H}'_{\text{opp}}} \Pr(S|\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}})}. \tag{5.1}$$

The second question is: how does it predict $\Pr(S|\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})$? The probability of all the public actions given the hidden information is the product of the probability of each public action given the hidden information i.e.

$$\Pr(S|\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) = \prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}). \tag{5.2}$$

Substituting Equation 5.2 into Equation 5.1 gives

$$\begin{aligned} \Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S) = \frac{\Pr(S|\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})}{\sum_{\mathcal{H}'_{\text{opp}}} \Pr(S|\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}})} = \\ \frac{\prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})}{\sum_{\mathcal{H}'_{\text{opp}}} \prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}}) \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}})}. \end{aligned} \tag{5.3}$$

Equation 5.3 can be simplified by cancelling out the agent's action probabilities, as these are the same for each possible instance of the opponent's hidden information. In turn, since the opponent's action probabilities only depend on its hidden information, then the agent's hidden information can be removed from the conditions. This leads to

$$\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S) = \frac{\prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}_{\text{opp}})^{b_i} \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})}{\sum_{\mathcal{H}'_{\text{opp}}} \prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}'_{\text{opp}})^{b_i} \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}})}$$
$$\text{where } b_i = \begin{cases} 1 & \text{if } a_i \text{ is an opponent action} \\ 0 & \text{otherwise} \end{cases}. \qquad (5.4)$$

The third question is: how does it calculate $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})$? The probability of the player and the opponent having particular instances of hidden information depends on the game. In die-roll poker, each six-sided die-roll is independent and $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) = \Pr(\mathcal{H}_{\text{pla}}) \Pr(\mathcal{H}_{\text{opp}}) = \frac{1}{6^{|\mathcal{H}_{\text{pla}}||\mathcal{H}_{\text{opp}}|}}$. In Rhode Island hold'em, each card draw is not independent as card draws are from the same fifty-two card deck and $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) = \frac{1}{|D|(|D|-1)}$ where $|D|$ is the size of the deck. For die-roll poker and Rhode Island hold'em, the joint probability of the players' hidden information is independent of what that hidden information is, meaning that $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}})$ would factor out in the denominator and cancel with $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})$ in the numerator of Equation 5.4. However, in general this is not the case. For example, in bucketed Rhode Island hold'em, if the public cards have a high squared expected hand strength, then the probability of each player's hand being in a high bucket sequence is higher, and if a player's hand is in a particular bucket sequence, then it is slightly less likely that the opponent's hand is in the same bucket sequence. For bucketed Rhode Island hold'em, in Equation 5.4, first substitute $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}}) = \Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}) \Pr(\mathcal{H}_{\text{pla}})$ and $\Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}}) = \Pr(\mathcal{H}'_{\text{opp}}|\mathcal{H}_{\text{pla}}) \Pr(\mathcal{H}_{\text{pla}})$ and since the agent's own hidden information is fixed then $\Pr(\mathcal{H}_{\text{pla}})$ can be factored out of the denominator and cancelled with the same term in the numerator. Calculating $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}})$ exactly can be done through enumeration. For bucketed Rhode Island hold'em, this involves counting how many times its hand is in the bucket sequence $\mathcal{H}_{\text{opp}}$, and dividing by the number of times its hand is in any bucket sequence.

The fourth question is: how does it predict $\Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}_{\text{opp}})$ where a_i is an opponent action? If a_i is an opponent action, then the opponent's hidden information $\mathcal{H}_{\text{opp}}$ and all previous public actions $(a_1, a_2, \ldots, a_{i-1})$ represent an opponent information set where the opponent acts $I = \{\mathcal{H}_{\text{opp}}, (a_1, a_2, \ldots, a_{i-1})\} \in \mathcal{I}_{\text{opp}}$ where $P(I) = \text{opp}$. It could use the sequence predictor p_I to predict $\Pr(a_i|I)$.

The problem with this is that it can create a sort of negative feedback loop. If the sequence predictor is inaccurate, which it probably will be initially, then its prediction of $\Pr(a_i|I)$ will be inaccurate, making the prediction of $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S)$ inaccurate, which will result in the wrong sequence predictors being updated, possibly making the next prediction of $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S)$ even more inaccurate. Instead of using sequence predictors, which cannot readily be partially updated by making fractional observations to account for uncertainty, this approach uses empirical probabilities, which can be partially updated to account for uncertainty.

Specifically, the EM component assumes that each discrete probability distribution $f_{A(I)}$ in the opponent's strategy $\sigma_{\text{opp}} = \{f_{A(I)} : I \in \mathcal{I}_{\text{opp}} \text{ and } P(I) = \text{opp}\}$ is a fixed categorical distribution. The parameters of $f_{A(I)}$ are the opponent's action probabilities at I. The idea is to set each parameter of each $f_{A(I)}$ to its maximum likelihood estimate given the observations. If samples from $f_{A(I)}$ could be observed, then maximising its parameters would be relatively easy. However, the number of times that the opponent has played action a in I, or the total number of actions that it has played in I, may be unknown. This is because if the opponent's hidden information is not observed, then the information sets that it played actions in are hidden. Thus, instead of observing samples from $f_{A(I)}$, samples are observed from a mixture of these categorical distributions, which include $f_{A(I)}$. In general, the maximum likelihood estimate for the probability of sampling category c from a categorical distribution d, given N samples from a mixture of K categorical distributions (including d) each with D categories, is the sum of the responsibilities of d to each c sample divided by the sum of the responsibilities of d to any sample i.e.

$$\begin{aligned}\mu_{dc} &= \frac{\sum_{n=1}^{N} \gamma(z_{nd}) x_{nc}}{\sum_{i=1}^{D} \sum_{n=1}^{N} \gamma(z_{nd}) x_{ni}} \\ \text{where } \gamma(z_{nd}) &= \frac{\pi_d \Pr(\vec{x}_n|\vec{\mu}_d)}{\sum_{j=1}^{K} \pi_j \Pr(\vec{x}_n|\vec{\mu}_j)},\end{aligned} \tag{5.5}$$

μ_{dc} is the probability of sampling category c from categorical distribution d, z_{nd} is the d-th component of the 1-of-K encoded vector $\vec{z}_n$, $\gamma(z_{nd})$ is the responsibility of d to sample n, x_{nc} is the c-th component of the 1-of-D encoded vector $\vec{x}_n$ and π_d is the probability of sampling from d. This is derived in Appendix A.2. Equation 5.5 can be used to set the parameters of the EM component's categorical distributions to their maximum likelihood estimates. In this case, π_d is the probability of having played into the opponent information set associated with d and $\Pr(\vec{x}_n|\vec{\mu}_d)$ is the probability of the opponent's action sampled from d. Thus, $\gamma(z_{nd})$ is equal to Equation 5.4

$$\gamma(z_{nd}) = \Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S) =$$
$$\frac{\prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}_{\text{opp}})^{b_i} \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}_{\text{opp}})}{\sum_{\mathcal{H}'_{\text{opp}}} \prod_{i=1}^{|S|} \Pr(a_i|(a_1, a_2, \ldots, a_{i-1}), \mathcal{H}'_{\text{opp}})^{b_i} \Pr(\mathcal{H}_{\text{pla}}, \mathcal{H}'_{\text{opp}})}$$
$$\text{where } b_i = \begin{cases} 1 & \text{if } a_i \text{ is an opponent action} \\ 0 & \text{otherwise} \end{cases}. \tag{5.6}$$

Iterative updates can be applied to μ_{dc} by rewriting Equation 5.5 as

$$\mu_{dc} = \frac{\left(\sum_{n=1}^{N-1} \gamma(z_{nd}) x_{nc}\right) + \gamma(z_{Nd}) x_{Nc}}{\left(\sum_{i=1}^{D} \sum_{n=1}^{N-1} \gamma(z_{nd}) x_{ni}\right) + \sum_{i=1}^{D} \gamma(z_{Nd}) x_{Ni}}. \tag{5.7}$$

A map from opponent information sets to real numbers $M_v : \mathcal{I}_{\text{opp}} \to \mathbb{R}$ can be used to store the numerator of Equation 5.5. For example, given a particular opponent information set $I \in \mathcal{I}_{\text{opp}}$ where the opponent acts $P(I) = \text{opp}$ the probability of sampling action $c \in A(I)$ from its categorical distribution $d = f_{A(I)}$ is

$$\mu_{dc} = \frac{\sum_{n=1}^{N} \gamma(z_{nd}) x_{nc}}{\sum_{i=1}^{D} \sum_{n=1}^{N} \gamma(z_{nd}) x_{ni}} = \frac{M_v((I, c))}{M_v(I)}. \tag{5.8}$$

The map M_v is the (expected) visit counts because the numerator of Equation 5.8 can be seen as the expected number of times action c is sampled from distribution d at opponent information set I, which in this case is the same as the expected number of times opponent information set (I, c) is visited. Likewise, the denominator of Equation 5.8 can be seen as the expected number of times any action is sampled from distribution d at opponent information set I, which in this case is the expected number of times opponent information set I is visited.

At the end of a game, let the opponent's reached terminal information set be $\{\mathcal{H}_{\text{opp}}, S\} \in \mathcal{I}_{\text{opp}}$. For each opponent information set $I \in \mathcal{I}_{\text{opp}}$ that the opponent could have acted at $P(I) = \text{opp}$, where $I = \{\mathcal{H}_{\text{opp}}, (a_1, a_2, \ldots, a_i)\}$ and $i < |S|$, update the parameters of the categorical distribution $d = f_{A(I)}$ associated with I using the action that the opponent could have sampled from it a_{i+1} as follows:

1. E-step: Calculate $\gamma(z_{nd})$ using Equation 5.6.

2. M-step: Update the parameters of d using Equation 5.7.

The opponent's hidden information h_{opp} can now be sampled from the categorical distribution $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S)$ computed in the E-step and used to update the relevant sequence predictors.

$$\begin{aligned}\text{Observe action } a_{i+1} \text{ with the sequence predictor}\\ M_{\text{pred}}(\{h_{\text{opp}}, (a_1, a_2, \ldots, a_i)\}) \text{ for all } 0 \leq i \leq |S|\\ \text{where } P(\{h_{\text{opp}}, (a_1, a_2, \ldots, a_i)\}) = \text{opp}.\end{aligned} \tag{5.9}$$

Here M_{pred} is a map from opponent information sets where the opponent acts to sequence predictors $M_{\text{pred}} : \{I : I \in \mathcal{I}_{\text{opp}} \text{ and } P(I) = \text{opp}\} \to \mathcal{P}$. Algorithm 8 shows the details of the opponent model.

5.5.2 Sequence Prediction in the Opponent Model

Whereas in Chapter 4 many sequence prediction methods were considered, here only ELPH by Jensen et al., 2005 is used to predict the opponent's actions. This is because rather than comparing sequence prediction methods, this chapter is looking to use any sequence prediction method to improve the agent's final performance. The main advantage of ELPH is that it can rapidly learn a non-stationary opponent strategy, which has allowed it to be used to defeat human and agent players in simple games [Jensen et al., 2005] and will allow it to be helpful against dynamic opponents. The opponent model creates a set of sequence predictors $\mathcal{P}$, which are instances of ELPH, one for each of the opponent's information sets where it acts $\mathcal{P} = \{p_I : I \in \mathcal{I}_{\text{opp}} \text{ and } P(I) = \text{opp}\}$. At the end of each game, the opponent's hidden information is sampled from a probability distribution, which is inferred using online expectation-maximisation. Using this information, the opponent information sets that the opponent acted at during the game are predicted and the sequence predictors associated with them observe the opponent's actions taken in them. Each instance of ELPH observes opponent actions in its associated opponent information set taken across different games. If a dynamic opponent changes this action distribution, then the ELPH instance will rapidly learn the new distribution from its set of observation-based hypothetical conditional distributions favouring those with low entropy and high predictability. The ELPH algorithm is shown in Algorithm 7 and includes the OBSERVE and

PREDICT functions called by the opponent model.

ELPH works by forming distributions conditioned on interaction histories of different lengths, pruning those with high entropies, and predicting using one with the minimum entropy. Given an observation, $s \in \Sigma$, it generates the set of all subsequences of its short-term memory, $\mathcal{P}(S)$, and for each subsequence creates or updates a distribution conditioned on it by incrementing the count for the subsequence and the observation in its long-term memory, $L(S', s) \leftarrow L(S', s) + 1$ for all $S' \in \mathcal{P}(S)$. It then prunes each conditional distribution (by removing its counts) if its normalised Shannon entropy, $\overline{H}$, is above a passed in threshold, H_l, for each $S' \in \mathcal{P}(S)$ $L \setminus (S', s)$ for all $s \in \Sigma$ if $\overline{H}(L(S')) > H_l$. Finally it adds the observation to the end of its short-term memory and removes the first observation if S is above its size-k limit. To make a prediction, it again gets the set of all subsequences of its short-term memory, $\mathcal{P}(S)$, and predicts using the distribution conditioned on one of these subsequences with the minimum reliable Shannon entropy, H_{rel}, $\arg\min_{S' \in \mathcal{P}(S)} H_{\text{rel}}(L(S'))$.

Algorithm 7 Entropy Learned Pruned Hypothesis Space (ELPH)

```
Require: Lookback (short-term memory size) $k \in \mathbb{N}^+$, entropy threshold $(0 \leq H_l \leq 1) \in \mathbb{R}$ and a set of symbols (possible observations) $\Sigma$.
1: Initialise short-term memory, which is a sequence of symbols $S \leftarrow ()$.
2: Initialise long-term memory, which is a map from sequences of symbols and symbols to counts $L : (s_1, s_2, \ldots, s_i) \times \Sigma \rightarrow \mathbb{N}^0$ where $0 \leq i \leq k$ and $s_i \in \Sigma$.
3: function Observe(a symbol $s \in \Sigma$)
4:     Get the set of all subsequences of $S$, $\mathcal{P}(S) \leftarrow \{(), (S(1)), \ldots, (S(|S|)), (S(1), S(2)), \ldots, (S(1), S(|S|)), \ldots, (S(1), S(2), \ldots, S(|S|))\}$.
5:     for all $S' \in \mathcal{P}(S)$ do
6:         if $(S', s) \notin L$ then
7:             Initialise the count of $s$ given $S'$, $L(S', s) \leftarrow 0$,
8:         end if
9:         Increment the count of $s$ given $S'$, $L(S', s) \leftarrow L(S', s) + 1$.
10:    end for
11:    if $-\frac{1}{\log|\Sigma|} \sum_{s' \in \Sigma} \frac{L(S',s')}{\sum_{s'' \in \Sigma} L(S',s'')} \log \frac{L(S',s')}{\sum_{s'' \in \Sigma} L(S',s'')} > H_l$ then ▷ High entropy.
12:        Remove all counts associated with $S'$, $L \setminus (S', s')$ for all $s' \in \Sigma$.
13:    end if
14:    Add observation to the end of $S$, $S \leftarrow (S(1), S(2), \ldots, S(|S|), s)$.
15:    if $|S| > k$ then
16:        Remove first (oldest) symbol from $S$, $S \leftarrow (S(2), S(3), \ldots, S(|S|))$.
17:    end if
18: end function
19: function Predict
20:    Get the set of all subsequences of $S$, $\mathcal{P}(S) \leftarrow \{(), (S(1)), \ldots, (S(|S|)), (S(1), S(2)), \ldots, (S(1), S(|S|)), \ldots, (S(1), S(2), \ldots, S(|S|))\}$.
21:    $S'' \leftarrow \arg\min_{S' \in \mathcal{P}(S)} -\frac{1}{\log(|\Sigma|+1)} \left( \frac{1}{1+\sum_{s \in \Sigma} L(S',s)} \log \frac{1}{1+\sum_{s \in \Sigma} L(S',s)} + \sum_{s \in \Sigma} \frac{L(S',s)}{1+\sum_{s' \in \Sigma} L(S',s')} \log \frac{L(S',s)}{1+\sum_{s' \in \Sigma} L(S',s')} \right)$. ▷ $S''$ has min adjusted entropy.
22:    return $\Pr(s|S'') = \frac{L(S'',s)}{\sum_{s' \in \Sigma} L(S'',s')}$ for all $s \in \Sigma$.
23: end function
```

5.5.3 Operation and Algorithm

The flowchart in Figure 5.1 shows how the overall approach operates. All steps are detailed in Algorithm 8 except for playing and simulating games as well as updating the agent's strategy. To play or simulate a game you simply start at the root of the game tree, sample and play an action according to the distribution of the player on turn, move to a new node according to that action and repeat the process until a terminal node is reached where rewards are assigned. The agent's strategy is updated using OS-MCCFR, which is explained in Section 3.2.9.

The time complexity of one iteration of the algorithm is dominated by the following (from most costly): 1. Simulating games. In each simulated game, at each non-terminal node, an action is sampled from a distribution, where a sequence predictor predicts each opponent distribution, and the agent updates its strategy using OS-MCCFR. This scales like $O(g[2^k d_{\max,\{\text{opp}\}} + d_{\max,\{\text{pla,cha}\}}]a_{\max})$ where g is simulated games, k is the lookback, $d_{\max,\text{N}}$ is maximum decisions in a game for players in N, and $a_{\max}$ is maximum actions at a node. 2. Sequence prediction. In general, a sequence predictor predicts using a number of distributions exponential in its lookback, which is the worst case for an ELPH instance. With a sequence predictor at each opponent information set where it acts predicting its distribution, this quickly becomes the bottleneck if the lookback grows faster than logarithmically with the game size. As shown above, this scales like $O(g2^k d_{\max,\text{opp}} a_{\max})$. 3. EM algorithm. After each game against the opponent it predicts probabilities and updates counts for each possible path. This scales like $O(d_{\max,\ \{\text{opp,pla,cha}\}}|\mathcal{H}_{\text{opp}}|)$ where $|\mathcal{H}_{\text{opp}}|$ is the number of opponent hidden information possibilities. 4. OS-MCCFR. After each game it updates regrets and probabilities at each of the agent's information sets where it acted. This scales like $O(gd_{\max,\{\text{pla}\}}a_{\max})$.

The space complexity of the algorithm is as follows. It stores regrets and probabilities for the actions for each of the agent's information sets where it acts, a sequence predictor for each opponent information set where the opponent acts, which has a number of distributions exponential in its lookback, and a count for each opponent information set. This scales like $O([|\mathcal{I}'_{\text{pla}}| + 2^k|\mathcal{I}'_{\text{opp}}|]a_{\max} + |\mathcal{I}_{\text{opp}}|)$ where $\mathcal{I}'_{\text{i}} = \{I : I \in \mathcal{I}_{\text{i}}, P(I) = \text{i}\}$.

The algorithm's efficiency mainly depends on game size. Larger games have more nodes, actions, and probably information sets, requiring more space, and more observations for EM and sequence prediction to learn to given overall accuracies. Exactly how time to converge/reach given overall accuracies or required lookback scales to larger games are open questions. Also, although overall regret after a number of OS-MCCFR iterations is bounded by theory Lanctot et al., 2009, how the algorithm affects this is an open question. To prevent bottlenecks, a large game may need an abstraction to reduce its size, and simulated games and lookback should be set sufficiently small.

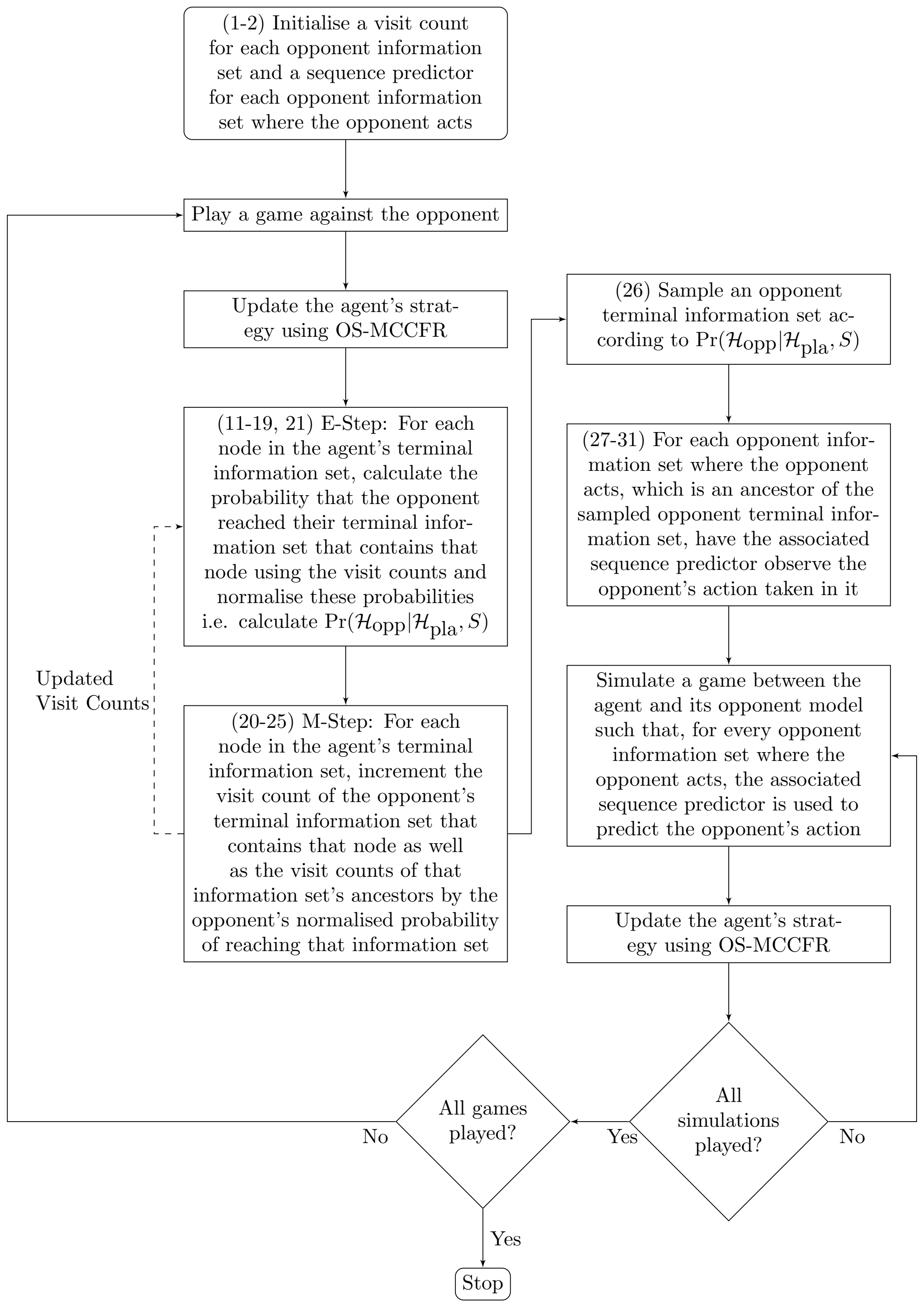

(1-2) Initialise a visit count for each opponent information set and a sequence predictor for each opponent information set where the opponent acts
Play a game against the opponent
Update the agent's strategy using OS-MCCFR
(11-19, 21) E-Step: For each node in the agent's terminal information set, calculate the probability that the opponent reached their terminal information set that contains that node using the visit counts and normalise these probabilities i.e. calculate $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S)$
Updated Visit Counts
(20-25) M-Step: For each node in the agent's terminal information set, increment the visit count of the opponent's terminal information set that contains that node as well as the visit counts of that information set's ancestors by the opponent's normalised probability of reaching that information set
(26) Sample an opponent terminal information set according to $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}}, S)$
(27-31) For each opponent information set where the opponent acts, which is an ancestor of the sampled opponent terminal information set, have the associated sequence predictor observe the opponent's action taken in it
Simulate a game between the agent and its opponent model such that, for every opponent information set where the opponent acts, the associated sequence predictor is used to predict the opponent's action
Update the agent's strategy using OS-MCCFR
All simulations played?
No
Yes
All games played?
No
Yes
Stop

Figure 5.1: A flowchart showing the steps of the approach. Bracketed numbers refer to line numbers in Algorithm 8 where the step is occurring. Note that the dashed line does not represent a flow of control, it highlights that the maximisation step influences the next expectation step by updating the visit counts.

Algorithm 8 Opponent Model using Sequence Prediction and EM

```
Require: Player information sets $\mathcal{I}_{\text{pla}}$, Opponent information sets $\mathcal{I}_{\text{opp}}$ and Lookback $k$.
 1: Initialise $M_v : \mathcal{I}_{\text{opp}} \to \mathbb{R}$ such that
    $M_v(I) \leftarrow \begin{cases} 1 & \text{if } I \subseteq Z, \\ \sum_{a \in A(I)} M_v((I,a)) & \text{otherwise.} \end{cases}$
 2: Initialise $M_{\text{pred}} : \{I : I \in \mathcal{I}_{\text{opp}}, P(I) = \text{opp}\} \to \mathcal{P}$        ▷ $\mathcal{P}$ is a set of sequence predictors.
 3: for all $I \in \text{dom}(M_{\text{pred}})$ do        ▷ $\text{dom}(f)$ is the domain of function $f$.
 4:     Initialise predictor $p_I$ with $k$ lookback $M_{\text{pred}}(I) \leftarrow p_I$.
 5:     for $i = 1$ to $k$ do
 6:         Sample action $a$ with probability $\frac{M_v((I,a))}{M_v(I)}$.
 7:         Observe action $a$ with $M_{\text{pred}}(I)$.
 8:     end for
 9: end for
10: function OBSERVE($(I' \subseteq Z) \in \mathcal{I}_{\text{pla}}$)
11:     Initialise $M_t : \{I : I \in \mathcal{I}_{\text{opp}}, I \cap I' \neq \emptyset\} \to \mathbb{R}$.
12:     for all $a_{1:m} \in \text{dom}(M_t)$ do        ▷ E-step, $a_{i:j} = (a_i, a_{i+1}, \ldots, a_j)$.
13:         $M_t(a_{1:m} = (\mathcal{H}_{\text{opp}}, S)) \leftarrow \Pr(\mathcal{H}_{\text{opp}} | \mathcal{H}_{\text{pla}})$.        ▷ $\Pr(\mathcal{H}_{\text{opp}} | \mathcal{H}_{\text{pla}})$ depends on the game.
14:         for $l \leftarrow 0$ to $m$ do
15:             if $P(a_{1:l}) = \text{opp}$ then
16:                 $M_t(a_{1:m}) \leftarrow M_t(a_{1:m}) \frac{M_v(a_{1:(l+1)})}{M_v(a_{1:l})}$.
17:             end if
18:         end for
19:     end for
20:     for all $a_{1:m} \in \text{dom}(M_t)$ do        ▷ M-step.
21:         $M_t(I) \leftarrow \frac{M_t(I)}{\sum_{I'' \in \text{dom}(M_t)} M_t(I'')}$.
22:         for $l \leftarrow 0$ to $m$ do
23:             $M_v(a_{1:l}) \leftarrow M_v(a_{1:l}) + M_t(a_{1:m})$.
24:         end for
25:     end for
26:     Sample $I = a_{1:m}$ with probability $M_t(I)$.
27:     for $l \leftarrow 0$ to $m$ do
28:         if $P(a_{1:l}) = \text{opp}$ then
29:             Observe action $a_{l+1}$ with $M_{\text{pred}}(a_{1:l})$.
30:         end if
31:     end for
32: end function
33: function PREDICT($h \in \{h : h \in H \setminus Z, P(h) = \text{opp}\}$)
34:     Get $I$ where $h \in I$ and $I \in \mathcal{I}_{\text{opp}}$.
35:     return a prediction from $M_{\text{pred}}(I)$.
36: end function
```

5.6 Evaluating the Three Proposals

The experiments measure the change in the average payoff per game of OS-MCCFR against several opponents in die-roll poker and Rhode Island hold'em if it plays simulated games against the opponent model in-between games against the actual opponent. Four variations of the opponent model are tested: 1. Without expectation-maximisation or sequence prediction (UN); 2. With just expectation-maximisation (EM); 3. With just sequence prediction (SP) and 4. With expectation-maximisation and sequence prediction (EM + SP); If expectation-maximisation is used, then it works as described in Section 5.5.1. Otherwise, the opponent's hidden information is sampled from $\Pr(\mathcal{H}_{\text{opp}}|\mathcal{H}_{\text{pla}})$. If sequence prediction is used, then ELPH predicts the opponent's actions as described in Section 5.5.2. Otherwise, the opponent's actions are predicted using empirical probabilities. The empirical probability of an action at an information set is the number of times it was played at that information set divided by the total number of actions played at that information set.

Table 5.1 shows all of the parameters used in the experiments. All players except CFRX and UCB share a standard small fixed exploration rate of 0.05 to ensure that they never stop exploring and that their exploration does not significantly affect their performances. Most other parameters are tuned to improve performances. For PGA-APP the result is that it uses a large learning rate of 0.9 to learn quickly, but also not too large as the environment is stochastic. It also uses a large discount factor of 0.99 to lookahead as far as possible. Its step-size is set to 0.01, large enough to learn quickly, but not too large as to constantly jump outside the strategy space. Its prediction length is set as in Zhang and Lesser, 2010. For UCB the result is that its constant is set to 3. Finally for OS-MCCFR OM it uses ELPH as its sequence predictor as it has been shown to perform well against human and agent players (see Chapter 4 and Jensen et al., 2005). Its lookback and numbers of simulated games are set as large as possible whilst still being tractable for these experiments and its entropy threshold is chosen to improve its tractability.

Table 5.1: Parameters used in the experiments.

Player	Parameters
CFRX	N/A
OS-MCCFR	explore rate = 0.05
PGA-APP	explore rate = 0.05, learning rate = 0.9, discount factor = 0.99, step-size = 0.01, prediction length = 1.0
UCB	constant = 3
OS-MCCFR OM	explore rate = 0.05, sequence predictor = ELPH (lookback = 5, entropy threshold = 0.1), games played against opponent model in-between games against the opponent = 100 in die-roll poker and 10 in Rhode Island hold'em
Other parameters	
number of games = 1×10^5, number of repeats = 80, both positions played per game	

5.6.1 Benefit of bucketed Rhode Island Hold'em

In Rhode Island hold'em players one and two have 2.50×10^7 and 2.46×10^7 information sets where they act respectively. This is too many for the agents used in this chapter to learn a high-reward strategy within 1×10^5 games, which is the number of games chosen to evaluate agents over in the experiments. This is because even if an agent updates its strategy at the maximum of 6 information sets per game (3 betting rounds $\times$ 2 decisions per betting round and player), then it would take more than 4.2×10^6 games to update each information set once. Thus making it impossible for an agent to learn a perfect strategy in Rhode Island hold'em within 1×10^5 games. Even learning an imperfect, but effective strategy, would probably require each information set to be visited many times. Learning an effective strategy within this number of games requires learning to be generalised across information sets using an abstraction.

To test the benefit of the abstraction the final average payoffs per game of OS-MCCFR, PGA-APP, and UCB were compared against a simple strategy that always raises in unabstracted and abstracted versions of Rhode Island hold'em. The abstraction reduces the number of information sets where each agents acts to 2.52×10^3 using percentile bucketing based on expected hand strength squared with five buckets for the pre-flop, flop, and turn stages in the game i.e. $b_1 = 5$, $b_2 = 5$ and $b_3 = 5$ (see Section 3.1.7). Figure 5.2 shows that each agent's final average payoff per game is negative in the unabstracted version and positive in the abstracted version. Thus, the abstraction allows each agent to learn to win against always raise. Linear least squares regression on the last 5×10^4 games in the unabstracted version estimates that it would take these agents 4.73×10^5, 1.22×10^6 and 1.69×10^6 games respectively to break even with a final average payoff per game of zero. These results show the benefit of abstraction in Rhode Island hold'em for agents to learn effective strategies within 1×10^5 games. From this point for Rhode Island hold'em agents use the bucketed version and are restricted to playing strategies within it. Better strategies likely exist in larger (finer) abstractions, but would take longer to learn. An agent might perform better with a smaller abstraction as it allows them to adapt faster.

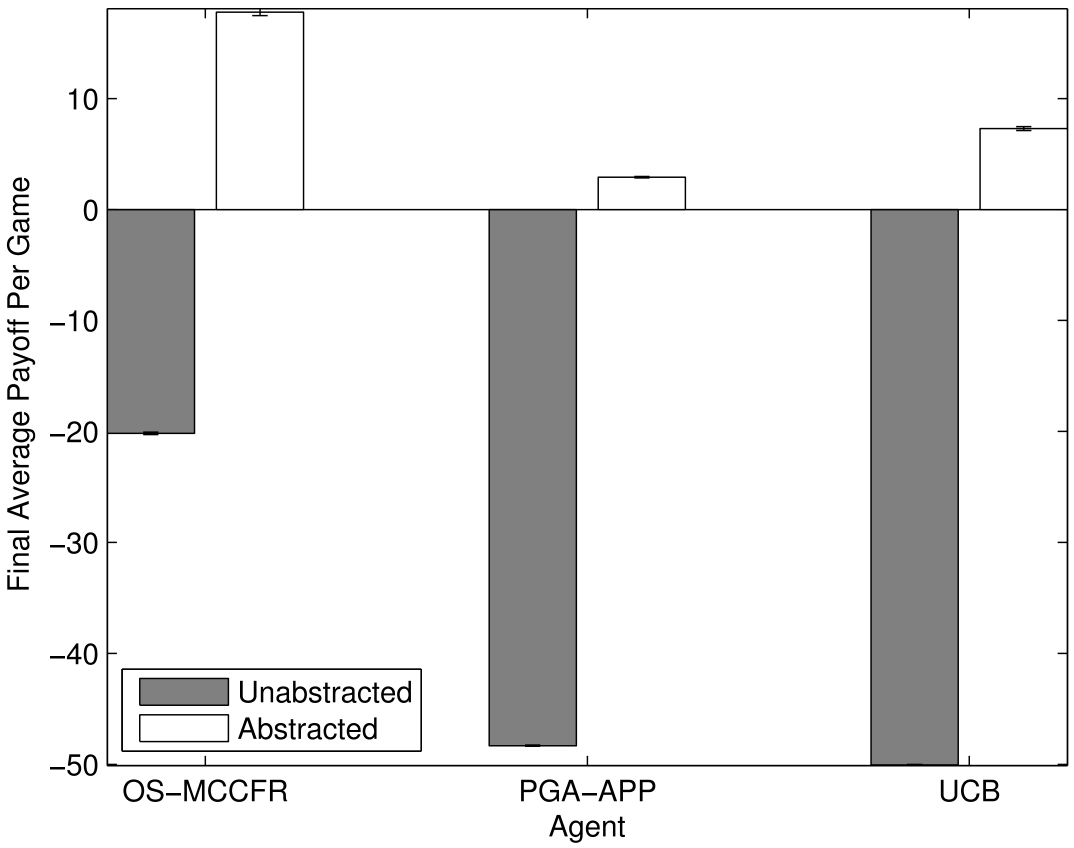


Figure 5.2: This bar chart shows the final average payoff per game of OS-MCCFR, PGA-APP, and UCB against a simple strategy that always raises in unabstracted and abstracted versions of Rhode Island hold'em. The abstracted version uses percentile bucketing based on expected hand strength squared with five buckets for the pre-flop, flop, and turn stages in the game.

5.6.2 Performance in Die-Roll Poker and Rhode Island Hold'em against the Opponents

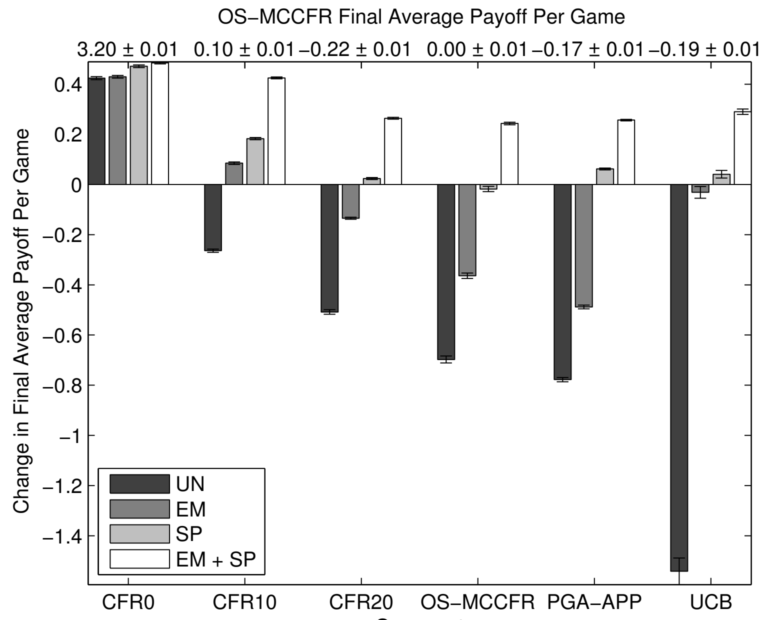
Figure 5.3 shows the change in the final average payoff per game of OS-MCCFR with the four variations of the opponent model. Firstly, it is always better with (EM) rather than (UN) except in die-roll poker against CFR0 because CFR0 plays actions uniformly at random and so its strategy does not depend on its hidden information. This supports the first secondary idea showing that inferences of the opponent's hidden information based on its behaviour using expectation-maximisation give higher final performances than inferences ignoring its behaviour. Secondly, it is always better with (SP) rather than (EM). This supports the second secondary idea showing that predictions of the opponent's actions using sequence prediction give higher final performances than predictions using empirical probabilities. Finally, it is always increased with (EM + SP), supporting the main idea that playing extra games between the agent and the opponent model improves the agent's final performance.

The results can be used to estimate how OS-MCCFR with each of the four variations of the opponent model will perform in the long term. To estimate long-term final performances exponential functions of the form $f(x) = ae^{-bx} + c$ were fitted to model average payoffs per game. Here $f(x)$ is the average payoff per game, x is the game number divided by 1×10^5 (the number of games) and a, b, and c are parameters. The c parameter is particularly interesting because it represents the asymptotic final average payoff per game. This also makes the time it takes to get close to c interesting. These estimated functions assume monotonically increasing or decreasing average payoffs per game for negative or positive values of the a parameter respectively. This may be true against an opponent with a stationary strategy as the average payoff per game of the approach is likely to monotonically increase as OS-MCCFR decreases overall regret in expectation per iteration against opponents with stationary strategies or in self-play [Lanctot et al., 2009]. However, against opponents with changing strategies this is not necessarily the case. For example, an opponent could constantly switch between a best-response strategy against the current strategy of the approach and a Nash equilibrium strategy until both are the same. On the other hand, in these experiments all opponents with changing strategies base them on accumulated statistics and so their strategies will eventually converge and become stationary.

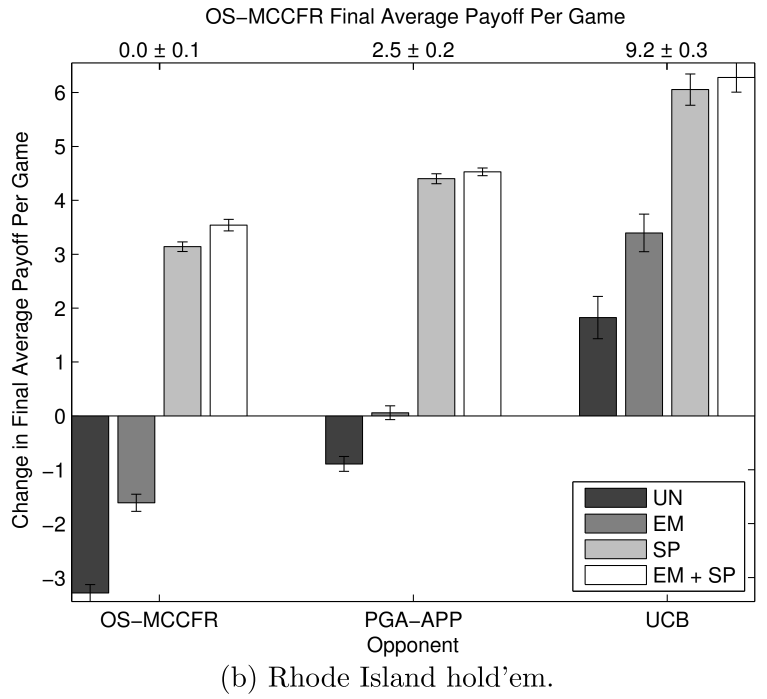
Thus, whilst these estimated functions may not be completely accurate, particularly for opponents with changing strategies in early iterations, they may still provide some useful insight into long-term final performances, particularly for opponents with stationary strategies or for later iterations.

These functions were fitted using MATLAB's Trust-Region-Reflective Least Squares algorithm with Bisquare weights, which is a non-linear least squares regression method found in its Curve Fitting Toolbox [*Least-Squares Algorithms*]. Tables 5.2 and 5.3 show the estimated parameters, including the c parameter and the estimated time it would take to reach 99% of c. Note that the time estimates are in terms of the number of iterations or games. In terms of absolute computation time, the approach generally takes longer per iteration than any of the opponents. Compared to OS-MCCFR this is easy to see as the approach uses the same algorithm, but also builds an opponent model and updates using rewards from simulated games. More specifically, OS-MCCFR, PGA-APP, and UCB only sample from and update their decision points encountered during a game. Thus, the time complexity of each one only scales with its maximum number of decision points per game, and its maximum number of actions per game at any of those decision points. Whereas for the approach this complexity is at least multiplied by the number of simulated games, but also has extra factors mainly due to sequence prediction and expectation-maximisation as outlined in Section 5.5.3. The c estimates reflect the results showing that (EM) is always better than (UN) (except against CFR0), (SP) is always better than (EM), and (EM + SP) always increases the average payoff per game. This indicates that this approach will continue to improve average payoffs per game in the long-term.

The final average payoff per game of (EM + SP) is not statistically significantly greater than that of (SP) in Rhode Island hold'em against PGA-APP and UCB. This could be because it takes longer to learn in Rhode Island hold'em as, firstly, even abstracted it has more information sets, and secondly, it has more hidden information ($5^3 = 125$ bucket sequences vs $6^2 = 36$ die rolls), which causes noisier play. In general, the EM component accuracy depends on the accuracy of its categorical distributions (one per opponent information set where it acts), so with more opponent information sets where it acts (due to more actions or hidden information) the more categorical distributions there will be, increasing learning time. To test this, the difference in the final average payoff per game between (EM + SP) and (SP) was tested against OS-MCCFR, PGA-APP, and UCB in die-roll poker with an increasing amount of hidden information (die faces). Table 5.5 shows that as the number of die faces is increased, the difference decreases. Additionally, expectation-maximisation is not helpful in Rhode Island hold'em if it infers that the opponent has the same bucket as the agent because this gives them both the same chance of winning.



(a) Die-roll poker.



(b) Rhode Island hold'em.

Figure 5.3: These bar charts show the change in the final average payoff per game of OS-MCCFR when used with variations of the opponent model including just expectation-maximisation (EM), just sequence prediction (SP), and both expectation-maximisation and sequence prediction (EM + SP). The results are shown for die-roll poker (Figure 5.3a) and Rhode Island hold'em (Figure 5.3b). The values are averaged over both positions and 80 repeats with the standard error of the mean shown at the top of each bar.

Table 5.2: Die-roll poker modelled average payoffs per game $f(x) = ae^{-bx} + c$ where x is the game number divided by 1×10^5 and $t_{0.99}$ is the time to reach 99% of c.

(a) Die-roll poker: CFR0.

Opponent model	a	b	c	$t_{0.99}$	Adjusted R-square
None	-1.2	3.1	3.2	1.2×10^5	0.99
UN	-0.54	4.4	3.6	6.2×10^4	0.99
EM	-0.52	4.2	3.6	6.4×10^4	0.99
SP	-0.24	3.5	3.7	5.3×10^4	0.99
EM + SP	-0.36	6.0	3.7	3.8×10^4	0.99

(b) Die-roll poker: CFR10.

Opponent model	a	b	c	$t_{0.99}$	Adjusted R-square
None	-0.85	2.6	0.15	2.4×10^5	0.99
UN	-0.64	2.7	-0.12	2.3×10^5	0.99
EM	-1.1	2.9	0.23	2.1×10^5	0.99
SP	-0.39	2.9	0.30	1.7×10^5	0.99
EM + SP	-0.56	3.7	0.53	1.2×10^5	0.99

(c) Die-roll poker: CFR20.

Opponent model	a	b	c	$t_{0.99}$	Adjusted R-square
None	-0.60	2.2	-0.16	2.7×10^5	0.99
UN	-0.74	3.1	-0.71	1.5×10^5	0.99
EM	-1.2	3.7	-0.36	1.6×10^5	0.99
SP	-0.48	3.4	-0.19	1.6×10^5	0.99
EM + SP	-0.40	3.5	0.05	1.9×10^5	0.99

Table 5.2

(d) Die-roll poker: OS-MCCFR.

Opponent model	a	b	c	$t_{0.99}$	Adjusted R-square
None	N/A	N/A	N/A	N/A	N/A
UN	0.47	6.1	-0.69	6.9×10^4	0.99
EM	1.5	28	-0.41	2.1×10^4	0.95
SP	0.45	2.5	-0.049	2.8×10^5	0.99
EM + SP	0.27	2.5	0.22	1.9×10^5	0.99

(e) Die-roll poker: PGA-APP.

Opponent model	a	b	c	$t_{0.99}$	Adjusted R-square
None	1.2	8.8	-0.18	7.4×10^4	0.99
UN	2.3	7.6	-0.96	7.2×10^4	0.99
EM	3.7	18	-0.71	3.5×10^4	0.98
SP	1.3	6.4	-0.1	1.1×10^5	0.99
EM + SP	1.9	8.8	0.085	8.7×10^4	0.97

(f) Die-roll poker: UCB.

Opponent model	a	b	c	$t_{0.99}$	Adjusted R-square
None	1.4	61	-0.24	1.0×10^4	0.84
UN	1.2	3	-1.8	1.4×10^5	0.99
EM	0.93	19	-0.22	3.3×10^4	0.99
SP	0.50	3.9	-0.15	1.5×10^5	0.97
EM + SP	0.35	4.1	0.10	1.4×10^5	0.99

Table 5.3: Rhode Island hold'em modelled average payoffs per game $f(x) = ae^{-bx} + c$ where x is the game number divided by 1×10^5 and $t_{0.99}$ is the time to reach 99% of c.

(a) Rhode Island hold'em: OS-MCCFR.

Opponent model	a	b	c	$t_{0.99}$	Adjusted R-square
None	N/A	N/A	N/A	N/A	N/A
UN	6.7	7.2	-3.2	7.4×10^4	0.99
EM	6.1	11	-1.4	5.6×10^4	0.99
SP	2.1	2.1	2.9	2.0×10^5	0.99
EM + SP	2.5	2.1	3.3	2.1×10^5	0.97

(b) Rhode Island hold'em: PGA-APP.

Opponent model	a	b	c	$t_{0.99}$	Adjusted R-square
None	5.0	2	1.8	2.8×10^5	0.95
UN	8	2.3	0.82	3.0×10^5	0.95
EM	7.2	2.4	1.9	2.5×10^5	0.94
SP	8.4	2.6	6.4	1.9×10^5	0.95
EM + SP	8.4	2.2	6.1	2.2×10^5	0.95

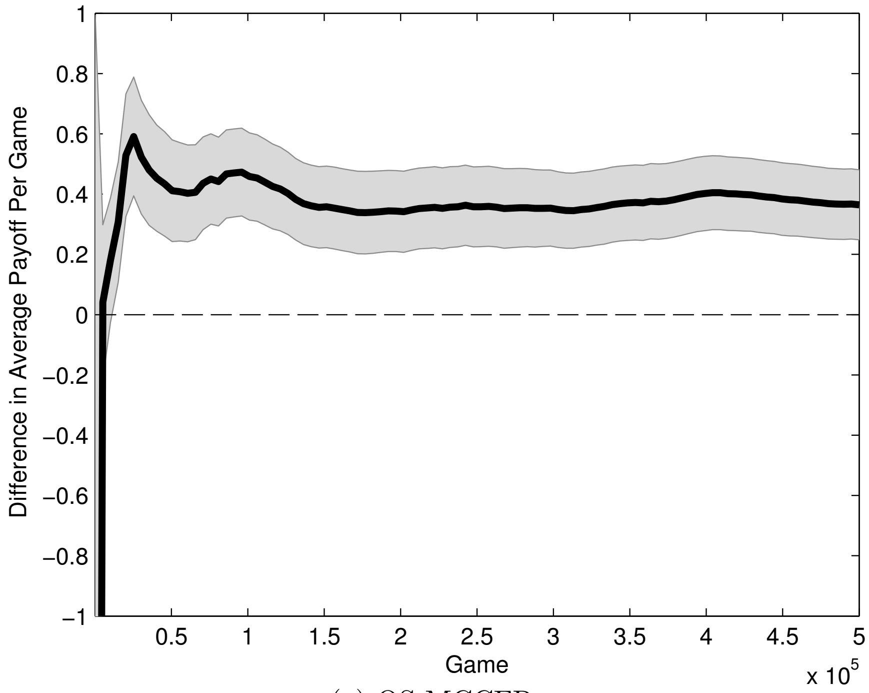
(c) Rhode Island hold'em: UCB.

Opponent model	a	b	c	$t_{0.99}$	Adjusted R-square
None	-4.4	1.9	9.8	2.0×10^5	0.98
UN	-3.4	2	12	1.7×10^5	0.94
EM	-5.5	1.6	14	2.3×10^5	0.95
SP	-3.8	2.1	16	1.5×10^5	0.96
EM + SP	-3.9	2.1	16	1.5×10^5	0.94

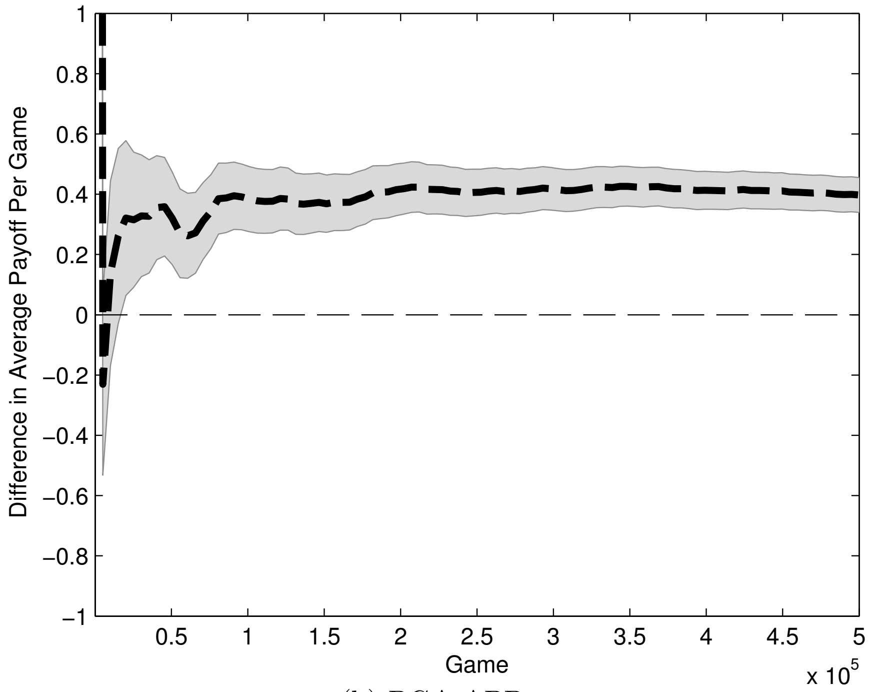
5.6.3 Further Experiments in Rhode Island hold'em

As previously mentioned, one problem with the results in Rhode Island hold'em is that, unlike those in die-roll poker, the final average payoff per game of (EM + SP) is not statistically significantly greater than that of (SP) against PGA-APP and UCB. This could be because it takes longer to learn in Rhode Island hold'em as, firstly, even abstracted it has more information sets, and secondly, it has more hidden information ($5^3 = 125$ bucket sequences vs $6^2 = 36$ die rolls), which causes noisier play. Various tests were performed to examine this.

The first test involved playing (EM + SP) and (SP) for more games in Rhode Island hold'em, 5×10^5 instead of 1×10^5. The idea being that an increased number of games would give agents more time to learn their strategies and the approach more time to learn the opponent's strategy. Figure 5.4 shows that after 5×10^5 games the final average payoff per game of (EM + SP) is statistically significantly higher than that of (SP) when playing against OS-MCCFR and PGA-APP but not against UCB. The differences against OS-MCCFR and PGA-APP appear to converge, whereas against UCB the difference seems to be levelling off but may not have converged. Finally, the differences against OS-MCCFR and PGA-APP remain fairly consistent after 1×10^5 games but the difference against UCB is still increasing. This only supports the idea that the agents need more time to learn when UCB is the opponent.

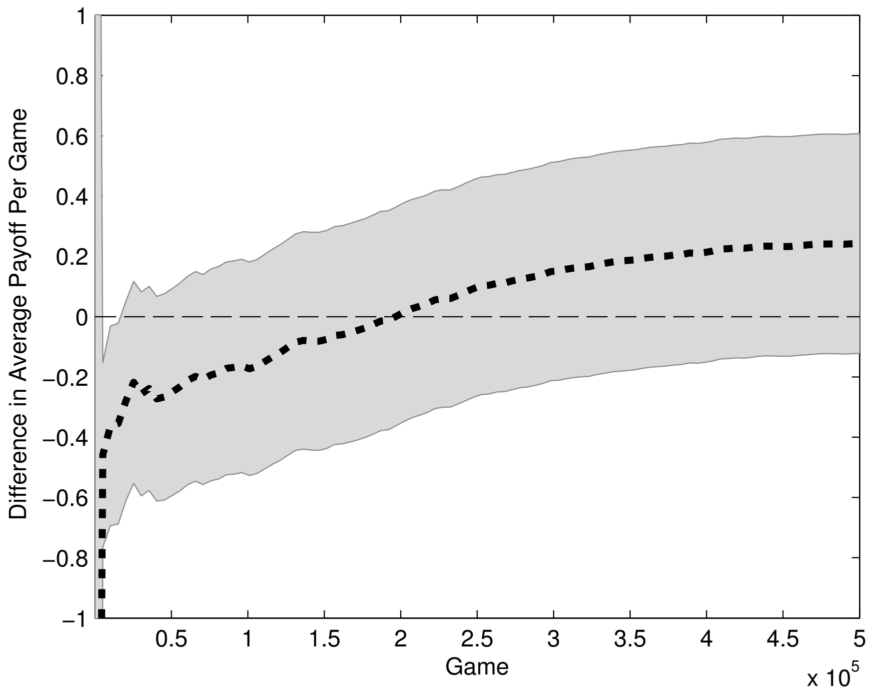


(a) OS-MCCFR.



(b) PGA-APP.

Figure 5.4



(c) UCB.

Figure 5.4: Each graph shows the average payoff per game of (EM + SP) - the average payoff per game of (SP) over 1×10^5 games in bucketed Rhode Island hold'em. The values are averaged over both positions and 80 repeats. The grey shaded area around each line represents the standard error of the mean.

The second test examined how similar each opponent behaves with different hidden information when played against (EM + SP). To do this, a distribution was formed consisting of an opponent's contribution to the probability of a sequence of public actions S given each instance of its hidden information $\mathcal{H}^i_{\text{opp}}$ and its strategy σ_{opp}, i.e. $P = (\Pr(S|\mathcal{H}^1_{\text{opp}}, \sigma_{\text{opp}}), \Pr(S|\mathcal{H}^2_{\text{opp}}, \sigma_{\text{opp}}), \dots)$. Following this, P was normalised, and the difference between it and a uniform distribution of the same size $D(P||Q)$ was measured where $Q = (\frac{1}{|P|}, \frac{1}{|P|}, \dots, \frac{1}{|P|})$. Any suitable distance metric could be used, and in this case the square root of the Jensen-Shannon divergence was used. Finally, the average of this distance over all sequences of public actions that lead to the end of the game was returned. Unfortunately, the calculation of this average distance is not ideal. Since the entire game tree cannot be stored in memory, only visited terminal information sets are considered. This probably overestimates the average distance as each unvisited terminal information set would probably have ancestors where the opponent acts that are also unvisited, and the opponent's strategy at those points would be the same as its initial strategy, which does not depend on its hidden information.

For two categorical distributions P and Q, the Kullback-Liebler divergence between P and Q is defined as

$$D_{KL}(P||Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)}. \tag{5.10}$$

The Kullback-Liebler divergence between P and Q is only defined if P and Q are valid categorical distributions and if $P(i) = 0$ whenever $Q(i) = 0$. Also, if $P(i) = Q(i) = 0$, then it is assumed that $0 \ln 0 = 0$. The Jensen-Shannon divergence between P and Q is defined as

$$D_{JS}(P||Q) = \frac{D_{KL}(P||M) + D_{KL}(Q||M)}{2} \text{ where } M = \frac{P+Q}{2}. \tag{5.11}$$

Here M is the "average" of P and Q, it is calculated by adding corresponding probabilities of P and Q and dividing each one by 2. Finally, the metric value $D_{JSM}(P||Q)$ is obtained by taking the square root giving

$$D_{JSM}(P||Q) = \sqrt{D_{JS}(P||Q)}. \tag{5.12}$$

The Kullback-Liebler divergence between P and Q has a minimum value of 0 and a maximum value of infinity. The Jensen-Shannon divergence between P and Q

has a minimum value of 0 and a maximum value of $\ln(2)$ if logarithms are to base e, or 1 if logarithms are to base 2. Thus, the Jensen-Shannon divergence metric has a minimum value of 0 and a maximum value of $\sqrt{\ln(2)}$ if logarithms are to base e, or 1 if logarithms are to base 2.

There are two main advantages to using the Jensen-Shannon divergence metric. Firstly, being based on the Kullback-Liebler divergence gives it a theoretical underpinning. Specifically, the Kullback-Liebler divergence is the average number of bits that are wasted by encoding events from the true distribution P with a code based on the estimated distribution Q. Secondly, it is a true metric that is always positive, symmetric, bounded, and well-defined, even with zero-values present in either distribution.

Algorithm 9 shows how to calculate this estimate. When calculating the Jensen-Shannon divergence metric, logarithms to the base 2 are used. Thus, the value returned is between 0 and 1 inclusive, and an average of these distances is also between 0 and 1 inclusive. If the estimate is 0, then it means that the opponent's strategy is completely independent of its hidden information. Whereas if the estimate is 1, then its behaviour is mainly dependent on its hidden information but it cannot be said to be completely dependent because its unvisited information sets are not considered. Table 5.4 shows these estimates for die-roll poker and Rhode Island hold'em after 1×10^5 games.

Algorithm 9 Estimating Strategy Dependence on Hidden Information

```
Require: Player $i$'s information sets $\mathcal{I}_i$, player $i$'s strategy $\sigma_i$, terminal nodes $Z$.
 1: Initialise estimate of player $i$'s hidden information dependence $d \leftarrow 0$.
 2: Initialise a count for the number of updates to the estimate $c \leftarrow 0$.
 3: for all $z \in Z$ do
 4:     $P \leftarrow ()$.
 5:     $Q \leftarrow ()$.
 6:     Get the public actions $S$ that lead to $z$.
 7:     for all $\mathcal{H}_i$ where $(S, \mathcal{H}_i) \in Z$ do
 8:         $P \leftarrow (P, \Pr(S|\mathcal{H}_i, \sigma_i))$.
 9:         $Q \leftarrow (Q, 1)$.
10:     end for
11:     Normalise $P$ and $Q$.
12:     $d \leftarrow d + D(P||Q)$              ▷ $D(P||Q)$ is distance between $P$ and $Q$.
13:     $c \leftarrow c + 1$
14: end for
15: return $\frac{d}{c}$.
```

Table 5.4: Estimated opponent strategy hidden information dependence after 1×10^5 games. A value of 0 means that its strategy is completely independent of its hidden information whereas a value of 1 means that its strategy is highly dependent on its hidden information. The values for Rhode Island hold'em are likely overestimated as unvisited information sets in it are not considered.

	Die-Roll Poker	Rhode Island hold'em
OS-MCCFR	0.72 ± 0.03	0.44 ± 0.01
PGA-APP	0.64 ± 0.07	0.26 ± 0.01
UCB	0.42 ± 0.03	0.35 ± 0.01

Table 5.4 shows that the estimate of the dependence of each opponent's strategy on its hidden information is higher in die-roll poker than in Rhode Island hold'em after 1×10^5 games. Also, the estimates in Rhode Island hold'em are likely inflated due to the calculation ignoring unvisited information sets. These results support the idea that the expectation-maximisation algorithm is less effective in Rhode Island hold'em than in die-roll poker because in Rhode Island hold'em the opponents' strategies are less dependent on their hidden information.

The third test looked at the difference in the final average payoff between (EM + SP) and (SP) against the opponents in die-roll poker with an increasing amount of hidden information (die faces). Table 5.5 shows that as the number of die faces is increased, the difference decreases. The reason for this is that as the amount of hidden information increases it becomes more likely that the opponent's behaviour will appear to be similar for multiple instances of hidden information. This makes it more difficult for expectation-maximisation to identify the opponent's hidden information based on its behaviour. These results support the idea that the expectation-maximisation algorithm is less effective as you increase the amount of hidden information, which also gives a reason for it being less effective in Rhode Island hold'em.

Table 5.5: Final average payoff per game of (EM + SP) - (SP) in die-roll poker.

	6 Die Faces	9 Die Faces	10 Die Faces
OS-MCCFR	0.26 ± 0.01	0.17 ± 0.01	0.12 ± 0.01
PGA-APP	0.19 ± 0.01	0.17 ± 0.01	0.09 ± 0.01
UCB	0.25 ± 0.02	0.24 ± 0.02	0.14 ± 0.02

Finally, expectation-maximisation is not helpful in Rhode Island hold'em if it infers that the opponent has the same bucket as the agent because this gives them both the same chance of winning.

5.7 Chapter Summary

This chapter has built on the work in Chapter 4 by using sequence prediction to model opponents in larger games that have more states and more hidden information. Specifically, three proposals were put forward for building and using an online opponent modelling algorithm that does not require knowledge outside the rules of the game, and does not assume that the opponent's strategy is stationary. Building the opponent model incorporates two proposals: an expectation-maximisation algorithm to infer the opponent's hidden information in an imperfect information game, and a sequence prediction method to specialise in predicting an opponent's changing strategy. Using the opponent model incorporates the third proposal: simulating games between the agent and the opponent model in-between games against the actual opponent. Experiments in simplified poker games show that this approach improves the average payoff per game of a state-of-the-art no-regret learning agent based on counterfactual regret minimisation. Future work will look at optimising the expectation-maximisation algorithm (e.g. tuning parameters like its step size), increasing the number of training games as the accuracy of the opponent model increases, and larger domains such as Texas hold'em poker, which may require improvements in scalability and further abstractions.

Chapter 6

Change Detection for Opponent Strategies

This chapter revisits the problem of modelling an opponent's strategy when it is changing. If an opponent's strategy is represented by a set of categorical distributions, then a change in its strategy means that at least one of these distributions changes. In this case the underlying problem would be modelling changing categorical distributions. A change in a categorical distribution shifts probability mass between two or more categories such that the resulting distribution is different and its probabilities still sum to one. In general, a change may occur for any arbitrary reason, depending on how the opponent works. However, for a rational opponent, a change is likely to occur because it tries to improve its strategy by learning from or reacting to its observations. A change may be sudden if, for example, the opponent switches its strategy, or it may be gradual if, for example, it modifies its action probabilities by a learning rate based on its rewards.

This chapter starts by outlining the main problems associated with modelling a changing distribution. It then describes variations of three state-of-the-art change detection methods, which include ADaptive WINdowing (ADWIN) by Bifet and Gavaldà, 2007, Bayesian Change Point Detection (BayesCPD) by Adams and MacKay, 2007 as well as Fearnhead and Liu, 2007, and Fisher's Exact Test Change Point Model (FET-CPM) by Ross, Tasoulis, and Adams, 2013. These methods are applied to model changing categorical distributions, which are representative of opponents with changing strategies. The overall idea is to represent the distribution accurately by keeping a window of observations from it, using one of these methods to detect when it changes, and discarding observations prior to detected change points. Ideally, this will result in a window that only contains observations that represent the current distribution.

The main contribution of this chapter is to experimentally compare the change detection methods against each other and against an empirical distribution at predicting changing categorical distributions. The results help determine the suitability of each method, relative to the other methods, for modelling opponents with changing strategies. The first comparison compares the accuracies of these methods at modelling categorical distributions that shift suddenly, or gradually, or a mixture of both. Whilst the second comparison compares the accuracies of these methods at modelling popular and state-of-the-art reinforcement learning agents as well as the rewards from playing best-response strategies against their models. The latter case has the added difficulty that the reinforcement learning agents are more likely to change their strategies more quickly as the models become more accurate. This is because more accurate models allow for more accurate best-response strategies with higher expected rewards, which in competitive games will lead to lower expected rewards for the agents being modelled causing them to "lose" more often and in turn to learn more quickly. For example, WoLF-PHC, WPL, and PGA-APP all have mechanisms to update their strategies more quickly if they think that they are losing.

In general, the results show that the change detection methods are more accurate than an empirical distribution at modelling changing categorical distributions and that they obtain higher rewards than the reinforcement learning agents. More specifically, the results show that the most accurate for small numbers of categories (3-5) is BayesCPD for sudden or mixed changes and ADWIN-D for gradual changes, whereas for large numbers of categories (10 or 20) BayesCPD-C is the most accurate for all types of changes. The results also show that increasing numbers of categories generally decreases accuracy, which makes sense because having more categories generally means each category has less probability mass making it harder to detect if probability mass is shifted between categories. Finally, the results show that against the reinforcement learning agents BayesCPD is generally the most accurate with the most rewarding best-response strategies.

6.1 Learning a Changing Distribution

Given an opponent has a finite and small number of actions at each of its decision points, then the goal of an opponent model can be seen as to learn, for each of its decision points, a categorical distribution over the associated actions from samples of them. This is challenging because there may not be many samples for a decision point due to infrequent visits, and also because its distribution may be changing over time. If its distribution is stationary, then the optimal estimate of the probability of a category is the ratio of the number of observations of it to the total number of observations. If its distribution is conditioned on a memory of previous states, or actions, or both, then it can be estimated using sequence prediction methods. Finally, if its distribution is changing, then some sequence prediction methods can account for this, but their approach is often to passively remove distributions that become unpredictive by some measure (e.g. ELPH removes distributions with high entropy). This chapter looks at change detection methods, which actively try to model changing distributions, as an alternative to, or as a compliment to, sequence prediction methods.

If the opponent changes one of its distributions over time, then the past samples from it may become unrepresentative. Ideally, the opponent model would detect when the distribution changes and discard unrepresentative past samples. One way to do this is to keep a window of representative observations. The size of the window is important. Ideally, the window should only keep samples that represent the current distribution. If the distribution is changing rapidly, then the window should be small, keeping only the most recent samples. Whereas if the distribution is changing slowly, then the window should be large, keeping many past samples. However, the smaller the window, the fewer samples there will be to estimate the distribution, making the estimate less accurate.

6.2 Adapting the Window Size

Maintaining a fixed-size window of past samples is a passive approach to modelling a changing distribution because the window size is set beforehand, but the ideal window size depends on how fast the distribution changes over time. Moreover, the distribution may not change at a fixed rate. There could be periods where it remains almost stationary, and other periods where it rapidly changes.

Another approach is to give older samples less weight (importance) than newer samples. This can be done by assigning new samples a maximum weight, which decays over time according to some function. However, this still has the problem of choosing a decay function to account for the unknown rate of change.

This section describes three state-of-the-art change detection methods, specifically ADaptive WINdowing (ADWIN) by Bifet and Gavaldà, 2007, Bayesian Change Point Detection (BayesCPD) by Fearnhead and Liu, 2007 as well as by Adams and MacKay, 2007, and Fisher's Exact Test Change Point Model (FET-CPM) by Ross, Tasoulis, and Adams, 2013. These methods can be seen as an active approach to modelling a changing distribution. Each one monitors observations received sequentially and attempt to detect when the underlying distribution generating them changes. If the most recent detected change point is accurate, then contracting the window to only include samples after it will ensure that the window only contains samples that represent the current distribution. It is assumed that the detected change points are accurate and so can be used in this way. These methods assume that the distribution undergoes abrupt changes, and that samples after one change point (or from the start) and before another change point are independently and identically distributed.

6.2.1 ADaptive WINdowing (ADWIN)

ADWIN by Bifet and Gavaldà, 2007 works by maintaining a sliding window of samples $W = (x_1, x_2, \ldots, x_t)$. Whenever two "large enough" subwindows of W show "distinct enough" means, a change is declared and the older portion of the window is discarded. A subwindow is defined as a contiguous part of the original window, i.e. a subwindow contains samples from index i to index j (inclusive) in the original window $W_{i:j} = (x_i, x_{i+1}, \ldots, x_j)$ where $1 \leq i \leq j$ and $i \leq j \leq t$. The window is not shortened as long as the null hypothesis, that the mean of the window has remained constant, is true with a confidence interval of $(1 - \delta)$. "Large enough" and "distinct enough" are made precise by choosing an appropriate statistical test. One advantage of ADWIN is that Bifet and Gavaldà, 2007 proved that it has rigorous performance guarantees by bounding the rates of false positives and false negatives. A false positive is detecting a change in the distribution when a change did not occur. A false negative is not detecting a change in the distribution when a change did occur. At each time step ADWIN does the following. Firstly, it adds the current sample, x_t, to its window $W \leftarrow$

(W, x_t). Secondly, for $i = 1$ to t, it checks if $|\mu_{W_{1:i}} - \mu_{W_{i+1:|W|}}| \geq \epsilon_i$, and if true for any i, then it stops, removes the first (oldest) sample from the window, and repeats this second step from $i = 1$. Here $\mu_{W_{i:j}}$ is the mean of the samples in the subwindow $W_{i:j}$ i.e. $\mu_{W_{i:j}} = \frac{\sum_{k=i}^{j} W(k)}{j-i+1}$, and ϵ_i is a threshold that depends on the split location i. The full algorithm for ADWIN is shown in Algorithm 10.

Algorithm 10 ADaptive WINdowing (ADWIN)

```
1: Initialise an empty window $W \leftarrow ()$.
2: for $t > 0$ do
3:     Observe the sample at time $t$, $x_t$.
4:     Add $x_t$ to the end of the window $W$, $W \leftarrow (W, x_t)$.
5:     compareSubWindows ← true.          ▷ Should subwindows be compared?
6:     while compareSubWindows = true do
7:         distinctSubWindows ← false.          ▷ Are subwindows distinct?
8:         for $i \leftarrow 1$ to $|W|$ do
9:             distinctSubWindows ← $|\mu_{W_{1:i}} - \mu_{W_{i+1:|W|}}| \geq \epsilon_i$.
10:            if distinctSubWindows = true then
11:                Remove the first (oldest) symbol from $W$, $W \leftarrow W_{2:|W|}$.
12:                break
13:            end if
14:        end for
15:        if distinctSubWindows = false then
16:            compareSubWindows ← false.
17:        end if
18:    end while
19: end for
```

The threshold ϵ_i is based on the Hoeffding bound, and is valid for all distributions. It is defined as

$$\epsilon_i = \sqrt{\left(\frac{1}{2i} + \frac{1}{2(|W| - i)}\right) \ln \frac{4|W|}{\delta}}. \tag{6.1}$$

However, according to Bifet and Gavaldà, 2007 it tends to overestimate the probability of large deviations for distributions with small variance as it assumes the worst-case variance of $\sigma^2 = 0.25$. They show ϵ_i can be tightened to

$$\epsilon_i = \sqrt{\left(\frac{2}{i} + \frac{2}{|W| - i}\right) \sigma_W^2 \ln \frac{2|W|}{\delta}} + \left(\frac{2}{3i} + \frac{2}{3(|W| - i)}\right) \ln \frac{2|W|}{\delta}. \tag{6.2}$$

Bifet and Gavaldà, 2007 assume that each sample x_t is between 0 and 1 (inclusive) i.e. $0 \leq x_t \leq 1$. However, even if x_t is between some arbitrary bounds a and b i.e. $a \leq x_t \leq b$, these bounds can be easily rescaled. Consequently, the implementations of ADWIN provided by Bifet and Gavaldà, 2007 assume that the samples from the distribution are either bits or real numbers, and that they have a mean value. ADWIN cannot be directly applied to a discrete categorical distribution because it does not have a mean value. There are many alternatives that could be used to measure the difference between these distributions. This chapter chooses to replace the mean comparison test in ADWIN with a measurement of the Jensen-Shannon divergence metric, which gives Algorithm 11.

Algorithm 11 ADaptive WINdowing using Distance (ADWIN-D)

```
1: Initialise an empty window $W \leftarrow ()$.
2: for $t > 0$ do
3:     Observe the sample at time $t$, $x_t$.
4:     Add $x_t$ to the end of the window $W$, $W \leftarrow (W, x_t)$.
5:     compareSubWindows ← true.          ▷ Should subwindows be compared?
6:     while compareSubWindows = true do
7:         distinctSubWindows ← false.               ▷ Are subwindows distinct?
8:         for $i \leftarrow 1$ to $|W|$ do
9:             distinctSubWindows ← $D_{JSM}(P_{W_{1:i}} || Q_{W_{i+1:|W|}}) \geq \gamma$.
10:            if distinctSubWindows = true then
11:                Remove the first (oldest) symbol from $W$, $W \leftarrow W_{2:|W|}$.
12:                break
13:            end if
14:        end for
15:        if distinctSubWindows = false then
16:            compareSubWindows ← false.
17:        end if
18:    end while
19: end for
```

In Algorithm 11, $P_{W_{i:j}}$ (or $Q_{W_{i:j}}$) represents an empirical categorical distribution formed over the subwindow $W_{i:j}$. Unfortunately, the same threshold ϵ_i cannot be used as in the original ADWIN Algorithm 10 because it was calculated for comparing means. Instead, a new threshold γ is used. This threshold could be set arbitrarily based on the range of the difference measure, which for the Jensen-Shannon divergence metric using logarithms to the base 2 is $[0, 1]$. However, it would be better if, like in the original algorithm, the threshold is set based on some theoretical foundation. The threshold could be based on the null hypothesis that elements of both windows are drawn from the same distribution. Dasu et al., 2006 proposed one way to do this. They predict a distribution over the value of the difference measure, assuming the null hypothesis is true. If the probability of the actual measured difference is significantly low enough, then they reject the null hypothesis. They predict the distribution over the difference measure using bootstrapping as follows. Given an empirical distribution over the first subwindow $P_{W_{1:i}}$, sample k multisets from it, each containing $2i$ elements $S1, S2, \ldots, Sk$. For each multiset, split it at i and measure the difference between the distributions over its first and second parts and add it to a set $S_d = \{D_{JSM}(P_{S1_{1:i}} || Q_{S1_{i+1:|S1|}}), D_{JSM}(P_{S2_{1:i}} || Q_{S2_{i+1:|S2|}}), \ldots, D_{JSM}(P_{Sk_{1:i}} || Q_{Sk_{i+1:|Sk|}})\}$. The distribution over the elements of S_d is then an estimate of the distribution over the difference measure. Finally, for a given confidence value α, if the actual measured difference is above the $(1 - \alpha)$-percentile of the distribution over S_d, then the null hypothesis is rejected.

The problem with this approach is its computational complexity. For every comparison between two subwindows, a number of bootstrap samples need to be taken to estimate the distribution over the difference measure, and thus calculate the significance of the actual measured difference. The exact number of bootstrap samples required depends on the distribution being sampled. However, Dasu et al., 2006 report that about 500 to 1000 samples work well. On top of this, most difference measures, such as the Jensen-Shannon divergence metric, will have a higher computational cost compared with calculating means. One reason for this is that, unlike other measures, means can be easily calculated iteratively.

Another approach is to associate a Bernoulli distribution with each category such that, observing that category would correspond to sampling a success from its distribution, and a failure from every other distribution. A window of samples from a categorical distribution with n categories could then be seen as n windows of samples from Bernoulli distributions. A window of Bernoulli samples for a category would contain a 1 (success) at every position it occurs in the window of categorical samples, and a 0 (failure) at every other position. The mean for a window of Bernoulli samples can be easily calculated as the number of successes (1s) divided by the number of successes and failures (1s and 0s). ADWIN can be applied to each category's window of Bernoulli samples separately. If any window is shortened, then all the other windows will be shortened. Technically, only the Bernoulli distributions for $n-1$ categories need to be checked for changes because, if this is false, then the Bernoulli distribution for the remaining category cannot have changed as there is no other category to shift probability mass from or to. However, if the number of categories is unknown beforehand, then the safest approach is to test all known categories. Although creating and monitoring a window for each category can be computationally expensive with many categories, in the games in this thesis the number of categories is always small. This is called ADWIN-B and is shown in Algorithm 12.

Algorithm 12 ADaptive WINdowing using Bernoulli distributions (ADWIN-B)

```
1: Initialise an empty window $W \leftarrow ()$.
2: Initialise an empty set $S \leftarrow \{\}$.
3: for $t > 0$ do
4:     Observe the sample at time $t$, $x_t$.
5:     Add $x_t$ to $W$, $W \leftarrow (W, x_t)$.
6:     if $x_t \notin S$ then          ▷ the sample type has not been observed before
7:         Add $x_t$ to $S$, $S \leftarrow S \cup \{x_t\}$.
8:     end if
9:     for each observed sample type $x \in S$ do
10:        Initialise an empty window for $x$, $W_x \leftarrow ()$.
11:        for $i \leftarrow 1$ to $|W|$ do
12:            if the sample in $W$ at $i$ is $x$, $W(i) = x$ then
13:                Set the symbol in $W_x$ at $i$ to 1, $W_x(i) = 1$,
14:            else
15:                Set the symbol in $W_x$ at $i$ to 0, $W_x(i) = 0$.
16:            end if
17:        end for
18:        compareSubWindows ← true.     ▷ Should subwindows be compared?
19:        while compareSubWindows = true do
20:            distinctSubWindows ← false.          ▷ Are subwindows distinct?
21:            for $i \leftarrow 1$ to $|W|$ do
22:                distinctSubWindows ← $|\mu_{W_{x(1:i)}} - \mu_{W_{x(i+1:|W_x|)}}| \geq \epsilon_i$.
23:                if distinctSubWindows = true then
24:                    Remove the first (oldest) sample from $W_x$, $W_x \leftarrow W_{x(2:|W_x|)}$.
25:                    Remove the first (oldest) sample from $W$, $W \leftarrow W_{2:|W|}$.
26:                end if
27:            end for
28:            if distinctSubWindows = false then
29:                compareSubWindows ← false.
30:            end if
31:        end while
32:    end for
33: end for
```

6.2.2 Bayesian Change Point Detection (BayesCPD)

Two different versions of this method are used. The first version, BayesCPD-B, associates a Bernoulli distribution with each category in the same way as ADWIN-B and FET-CPM-B, and so uses a beta distribution as its conjugate prior. The second version, BayesCPD-C, simply models the distribution as a categorical distribution, and so uses a Dirichlet distribution as its conjugate prior. This method works by calculating a posterior distribution over the runlength, and then using it to estimate the sample distribution [Adams and MacKay, 2007; Fearnhead and Liu, 2007]. In particular, it assumes that the sample distribution, conditioned on a particular run length, can be computed. This allows the marginal sample distribution to be found by integrating over its posterior distribution conditioned on the current run length

$$\Pr(x_{t+1}|x_{1:t}) = \sum_{r_t} \Pr(x_{t+1}|r_t, x_t^{(r)}) \Pr(r_t|x_{1:t}).$$

Here r_t is the runlength at time t, x_t is the sample at time t, $x_t^{(r)} = x_{t-r_t+1:t}$ is the set of samples associated with runlength r_t, and $x_{i:j}$ are the samples from time i to time j inclusive. The runlength is defined as the number of steps since the distribution last changed. To predict the last change point optimally, this method considers all possible runlengths and weights them by their probabilities given the samples. The authors show that exact inference on the runlength can be achieved using a message passing algorithm. The inference procedure is as follows

$$\begin{aligned}\Pr(r_t|x_{1:t}) &= \frac{\Pr(r_t, x_{1:t})}{\Pr(x_{1:t})}\\ &= \frac{\sum_{r_{t-1}} \Pr(r_t, r_{t-1}, x_{1:t})}{\Pr(x_{1:t})}\\ &= \frac{\sum_{r_{t-1}} \Pr(r_t, x_t|r_{t-1}, x_{1:t-1}) \Pr(r_{t-1}, x_{1:t-1})}{\Pr(x_{1:t})}\\ &= \frac{\sum_{r_{t-1}} \Pr(r_t|r_{t-1}) \Pr(x_t|r_{t-1}, x_t^{(r)}) \Pr(r_{t-1}, x_{1:t-1})}{\Pr(x_{1:t})}.\end{aligned}$$

Note that the sample distribution $\Pr(x_t|r_{t-1}, x_t^{(r)})$ is determined by the most recent data $x_t^{(r)}$. The last line assumes that the runlength is independent of the previous samples and only depends on the previous runlength $\Pr(r_t|r_{t-1}, x_{1:t-1}) = \Pr(r_t|r_{t-1})$. This is a message passing algorithm because r_t can only take values based on r_{t-1}. Specifically, either $r_t = 0$ if a change occurs, or $r_t = r_{t-1} + 1$ if a change does not occur. The probability $\Pr(r_t|r_{t-1})$ is given by a "hazard" function, $h(t)$, for both values. A simple approach is to assume that the hazard function returns a constant probability for a change occurring $\Pr(r_t = 0|r_{t-1}) = h(0) = \gamma$. The probability for a change not occurring would then be one minus the probability for a change occurring $\Pr(r_t = r_{t-1}+1|r_{t-1}) = h(r_{t-1}+1) = 1-\gamma$. The hazard function returns zero for all other values of t.

Although the assumption that there is a constant probability of a change point occurring may be incorrect, this parameter does not need to be specified exactly. If there is enough evidence that a change has occurred, then the algorithm is designed to detect it [Adams and MacKay, 2007; Fearnhead and Liu, 2007]. This parameter is essentially a trade-off between the detection delay and the amount of false positives. If the probability of a change point occurring is set high, then the detection delay will be shorter, but there will be more false positives. If the probability of a change point occurring is set low, then the detection delay will be longer, but there will be less false positives. Wilson, Nassar, and Gold, 2010 as well as Turner, Saatci, and Rasmussen, 2009 have proposed methods to learn the hazard function from the data. The method of Wilson, Nassar, and Gold, 2010 can learn a hazard function that is piecewise constant using a hierarchical generative model, whilst the method of Turner, Saatci, and Rasmussen, 2009 can learn any parametric hazard rate via gradient descent.

The space complexity grows linearly with the number of samples because there is a possible runlength for each sample. The time complexity also grows linearly because each possible runlength requires an update. To place an upper limit on the number of possible runlengths, and in turn these complexities, a particle filter is used as suggested by Fearnhead and Liu, 2007, which maintains a finite sample of the runlength distribution. A particle filter is a Monte Carlo method that estimates a sequential Bayesian model. Each particle represents a point in the distribution with its weight being its approximate probability. If the number of particles grows too large, then resampling takes place where some particles are thrown away and the weights of the remaining particles are updated. The resampling scheme used is called Stratified Optimal Resampling

(SOR). Under this scheme the reweighting ensures that the expected values of the new weights are the original weights. SOR is optimal in that the expected squared difference between the original weights and the new weights is minimised. The SOR procedure is shown in Algorithm 13.

Algorithm 13 Stratified Optimal Resampling (SOR)

```
Require: A set of particle probabilities $P \leftarrow \{p_1, p_2, \dots, p_{|P|}\}$, a desired number
    of particles $m \in \mathbb{N}^+$.
 1: Initialise a new set of particle probabilities $Q \leftarrow \{q_1, q_2, \dots, q_{|P|}\}$.
 2: Find $c$ such that it solves the equation $m = \sum_{i=1}^{|P|} \min(1, \frac{p_i}{c})$.
 3: Sample $u$ from a uniform distribution between 0 and $c$, $u \sim U(0, c)$.
 4: for all $p_i \in P$ do
 5:     if $p_i \geq c$ then
 6:         $q_i \leftarrow p_i$,
 7:     else
 8:         $u \leftarrow u - p_i$.
 9:         if $u \leq 0$ then
10:             $q_i \leftarrow c$,
11:             $u \leftarrow u + c$,
12:         else
13:             $q_i \leftarrow 0$.
14:         end if
15:     end if
16: end for
17: Discard particles in $Q$ with zero probability $Q \leftarrow Q \setminus q_i$ for all $q_i = 0$.
18: return $Q$.
```

6.2.3 Fisher's Exact Test Change Point Model (FET-CPM)

Fisher's Exact Test Change Point Model (FET-CPM) by Ross, Tasoulis, and Adams, 2013 is a frequentist change detection method that is designed to detect changes in the mean of an equally-spaced, discreet-time sequence of Bernoulli random variables. Like ADWIN and BayesCPM, it assumes that the mean is constant, but unknown, after one change point (or from the start) and before another change point. Its goal is to detect changes as soon after they occur as possible, as well as to estimate their locations. Ross, Tasoulis, and Adams, 2013 show that it performs comparably to the optimal CUMulative SUM Control Chart (CUSUM) scheme (a change detection method developed by Page, 1954) with knowledge of the pre-change and post-change parameter values. The approach uses a two-sample hypothesis test to infer if a change has occurred at a specific point with the null hypothesis being that no change has occurred. The main advantages of FET are that its null distribution can be computed exactly (without relying on asymptotic distributions) and it does not depend on the true value of the mean.

The method works as follows, given a sequence of t observations, $x_1, x_2, \ldots, x_t$, it is split into two samples, $x_1, x_2, \ldots, x_i$ and $x_{i+1}, x_{i+2}, \ldots, x_t$. The null hypothesis says that there has been no change point, meaning that both samples have been generated by the same Bernoulli distribution with the same mean. Table 6.1 shows the data. Fisher showed that the probability of obtaining any such set of

Table 6.1: Data from two Bernoulli samples.

	Sample 1 $(x_1, x_2, \ldots, x_i)$	Sample 2 $(x_{i+1}, x_{i+2}, \ldots, x_t)$	Sum
Successes	s_i	$s_t - s_i$	s_t
Failures	$i - s_i$	$(t-i) - (s_t - s_i)$	$t - s_t$
Sum	i	$t - i$	t

values, given the null hypothesis is true, follows a hypergeometric distribution. In this case giving

$$\Pr(s_i|s_t) = \frac{\binom{s_t}{s_i}\binom{t-s_t}{i-s_i}}{\binom{t}{i}}.$$

This probability is calculated for each value of i, and if the minimum of these probabilities is less than some threshold, then a change is assumed to have occurred at that point. The threshold for each time step is computed offline as follows. Numerous streams are simulated, each containing a number of observations sampled from a Bernoulli distribution with a mean of 0.5. For each stream, $(x_1, x_2, \ldots, x_t)$, and for each index within the stream, $1 \leq i \leq t$, the above probability is calculated for all observations between index 1 and i (inclusive). The threshold for $t = 1$ is found by taking the probabilities for $i = 1$ from all of the streams, and setting the threshold such that a fixed proportion, α, of these probabilities exceed it. The streams with probabilities exceeding this first threshold are then discarded and the second threshold is calculated by taking the probabilities for $i = 2$ from all of the streams, and setting the threshold such that α of these probabilities exceed it. This is repeated for all indexes. The process is slightly more involved than this description since Ross et al. also smooth the probabilities (using exponential smoothing) to reduce the discreetness of the test statistic. For exact details see Ross, Tasoulis, and Adams, 2013.

Similarly to ADWIN-B, a Bernoulli distribution is associated with each category, and this method is used to detect changes in any of them. When a change is detected in any window, all samples prior to that change point are discarded. Algorithm 14 shows how to apply their method using this approach.

Algorithm 14 Fisher's Exact Test Change Point Model using Bernoulli distributions (FET-CPM-B)

```
1: Initialise an empty window $W \leftarrow ()$.
2: Initialise an empty set $S \leftarrow \{\}$.
3: Initialise a map from samples to change point models $M : S \rightarrow$ FET-CPM.
4: for $t > 0$ do
5:     Observe the sample at time $t$, $x_t$.
6:     Add $x_t$ to $W$, $W \leftarrow (W, x_t)$.
7:     if $x_t \notin S$ then          ▷ The sample type has not been observed before.
8:         Add $x_t$ to $S$, $S \leftarrow S \cup \{x_t\}$.
9:         Map $x_t$ to a new FET-CPM $M(x_t) \leftarrow$ FET-CPM$_x$.
10:    end if
11:    for each observed sample type $x \in S$ do
12:        if $x$ is equal to $x_t$, $x = x_t$ then
13:            Have $M(x)$ observe 1 (success),
14:        else
15:            Have $M(x)$ observe 0 (fail).
16:        end if
17:    end for
18:    for each observed sample type $x \in S$ do
19:        if $M(x)$ has detected a change then
20:            Get the most likely change point $i$.
21:            Discard all symbols in $W$ before the change point $i$, $W \leftarrow W_{i:|W|}$.
22:            break
23:        end if
24:    end for
25: end for
```

6.3 Comparing Change Detection Methods

In this section, experiments are performed to compare the change detection methods in Section 6.2 to each other, and to an empirical distribution. The aim is to compare change detection methods at modelling changing distributions as well as changing opponent strategies. Section 6.3.1 compares the accuracies of these modelling algorithms at modelling changing categorical distributions, which can be seen as being representative of changing opponent strategies. Section 6.3.2 takes a more direct approach by using these modelling algorithms to play against popular and state-of-the-art reinforcement learning agents. Here the accuracies of these modelling algorithms are compared as well as the rewards received from playing best-response strategies to their predictions. Within these experiments, the empirical distribution is used as a baseline and simply estimates the probability of a symbol as the number of times that symbol has been observed divided by the total number of observations. The three state-of-the-art change detection methods, which are the basis for the other change detection methods in this chapter, have empirical results in their defining papers demonstrating their effectiveness, but this appears to be the first time a comparison has been done between them.

Bifet and Gavaldà, 2007 compare ADWIN to a sliding window, and to a flushing window. A flushing window uses a fixed window and a sliding window. The fixed window contains samples immediately after the most recent inferred change point, and the sliding window contains the most recent samples. The flushing window tests for changes between its two windows using the same test as ADWIN, and if a change is detected, then the samples in the fixed window are discarded, and the samples in the sliding window are moved into the fixed window, emptying the sliding window. Bifet and Gavaldà, 2007 also compare ADWIN to the change detection method of Gama et al., 2004. BayesCPD is used by Adams and MacKay, 2007 as well as Fearnhead and Liu, 2007 on several real-world datasets, but they do not compare it to other change detection methods. Ross, Tasoulis, and Adams, 2013 compare FET-CPM to the optimal CUSUM chart following the implementation of Marion R. Reynolds and Stoumbos, 1999.

According to Ross, Tasoulis, and Adams, 2013, the performance of a sequential change detector is often measured by two criteria, which are its expected time between false positive detections, and its mean delay until a change is detected. In this work, the only interest is in using these methods to help maintain accurate estimated distributions, and so the performance of each method is measured through its accuracy. In sections 6.3.1 and 6.3.2, accuracy is measured as the Jensen-Shannon divergence metric between the estimated and actual distributions. The best method is the one that maintains the lowest average distance over time between its estimated distribution and the actual distribution. In section 6.3.2, the number of correct predictions and the rewards are also provided, the former giving an additional, more direct, accuracy measure, and the latter showing the benefits of these accuracies.

For ADWIN and ADWIN-B, the confidence δ is used to calculate the thresholds, ϵ_i, which depend on the split location i. For ADWIN-D, γ is a threshold for the Jensen-Shannon divergence metric. A fixed γ is used, rather than determining it through bootstrapping, to reduce the computational complexity. A minimum window size can also be set to mitigate the inaccuracies of very small windows for categorical distributions, and a maximum window size to further reduce the computational complexity. For BayesCPD-B and BayesCPD-C, each use their own hazard function, $h(0)$, that returns a constant probability estimate for a change occurring, which does not match the actual probability for a change occurring. Additionally, each uses a number of particles to approximate the distribution and reduce the computational complexity. For empirical probability, there are no parameters. Finally, for FET-CPM-B, the average run length, ARL0, is an estimate of the average number of observations before a false positive occurs, assuming that the sequence does not change, the startup is the number of observations after which monitoring begins, and λ is a smoothing parameter, where smaller values are better at detecting smaller shifts and vice versa.

There is a tradeoff when setting the parameters δ, $(1-\gamma)$, $h(0)$, and 1/ARL0 in that low values reduce the number of false positives, but high values detect changes more quickly. In Section 6.3.1, these parameters are set based on initial experiments and the change detection methods' defining papers, but it is acknowledged that fine tuning may improve accuracy. Whereas in Section 6.3.2, these parameters are tuned to detect changes more quickly and to be more competitive against learning agents.

6.3.1 Modelling Changing Categorical Distributions

In each of these experiments, a sample is drawn from a categorical distribution at each time step. The change detection methods observe and use these samples to model the distribution, which may change either suddenly or gradually. A sudden change immediately replaces the distribution with a new categorical distribution. A gradual change incrementally modifies the distribution. Two types of gradual change are looked at. The first type of gradual change modifies the probability of each category by $(q-p)/t$ at each time step, where q is the new probability, p is the original probability, and t is the time for the gradual change to complete. In this case, the new probabilities are from a new categorical distribution. The second type of gradual change performs a random walk on the $(k-1)$-simplex, which represents the space of all categorical distributions with k categories.

For either a sudden change, or a gradual change of the first type, a new categorical distribution is sampled from a Dirichlet distribution with a vector of ones for its concentration parameter, which makes every categorical distribution of the same dimension as the concentration parameter equally likely to be sampled. For a gradual change of the second type, it starts by storing k samples from a unit-exponential distribution $(x_1, x_2, \ldots, x_k)$ where $x_i \sim e^{-x}$ for $1 \leq i \leq k$. The initial categorical distribution is calculated by normalising these samples. The unnormalised samples are kept, and at each time step, each unnormalised sample x_i is updated (walked) using Algorithm 15, which was first outlined by Fernandes and Atchley, 2008 and further explained in Fernandes, 2008. A new categorical distribution is calculated at each time step by normalising the updated unnormalised samples. Algorithm 15 uses the Metropolis-Hastings algorithm to perform the random walk on each unit-exponential variable. For a derivation of Algorithm 15 see Appendix C.

Algorithm 15 Metropolis-Hastings Unit-Exponential Sample

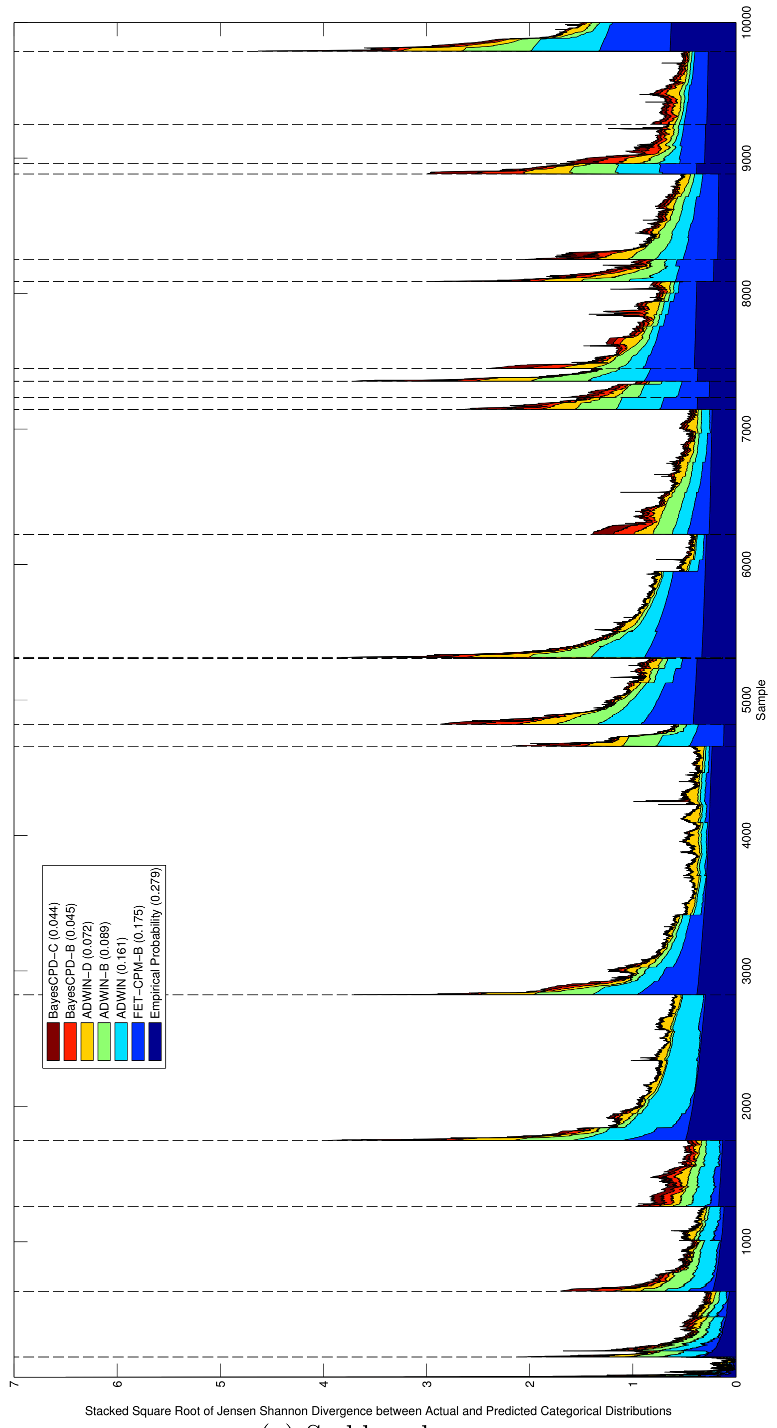
```
Require: Current point $x_{\text{old}}$, stepsize $h$.
1: Sample a new point $x_{\text{new}} \leftarrow x_{\text{old}} e^Z$ where $Z \sim \mathcal{N}(0, h)$.
2: Calculate Metropolis ratio $r_m \leftarrow \frac{e^{-x_{\text{new}}}}{e^{-x_{\text{old}}}}$.
3: Calculate Hastings ratio $r_h \leftarrow \frac{x_{\text{new}}}{x_{\text{old}}}$.
4: Calculate acceptance probability $p_{\text{acc}} \leftarrow \min(1, r_m r_h)$.
5: if $p_{\text{acc}} > u$ where $u \sim \mathcal{U}(0,1)$ then
6:     return $x_{\text{new}}$,
7: else
8:     return $x_{\text{old}}$.
9: end if
```

The parameters for the methods are as follows. The confidence δ is set to $\delta = 0.1$, which is a typical value used in Bifet and Gavaldà, 2007. The threshold γ is set to $\gamma = 0.3$, and each hazard function $h(0)$ is set to $h(0) = 1 \times 10^{-4}$, which are based on initial experiments. The average run length, ARL0, is chosen from a set of allowed values, and is set to the lowest of these values to respond to changes more quickly, startup is also set to its lowest value to respond to changes more quickly, and λ is set to its recommended value from one of two allowed values, these values can be found in Ross, 2013; Ross, Tasoulis, and Adams, 2013. For each experiment, the number of categories needs to be chosen. The probability of a change occurring for sudden changes, and gradual changes of the first type, is set to 2×10^{-3}. For gradual changes of the second type, the stepsize is set to $h = 0.05$. The length of a sudden change is set to 1, and the length of a gradual change of the first type is set to the time between the predetermined change points. Finally, for a mixed change, there is an equal probability of a sudden change, or a gradual change of the first type, or a gradual change of the second type occurring. All parameters are summarised in Table 6.2.

Table 6.2: Methods vs changing distributions parameters.

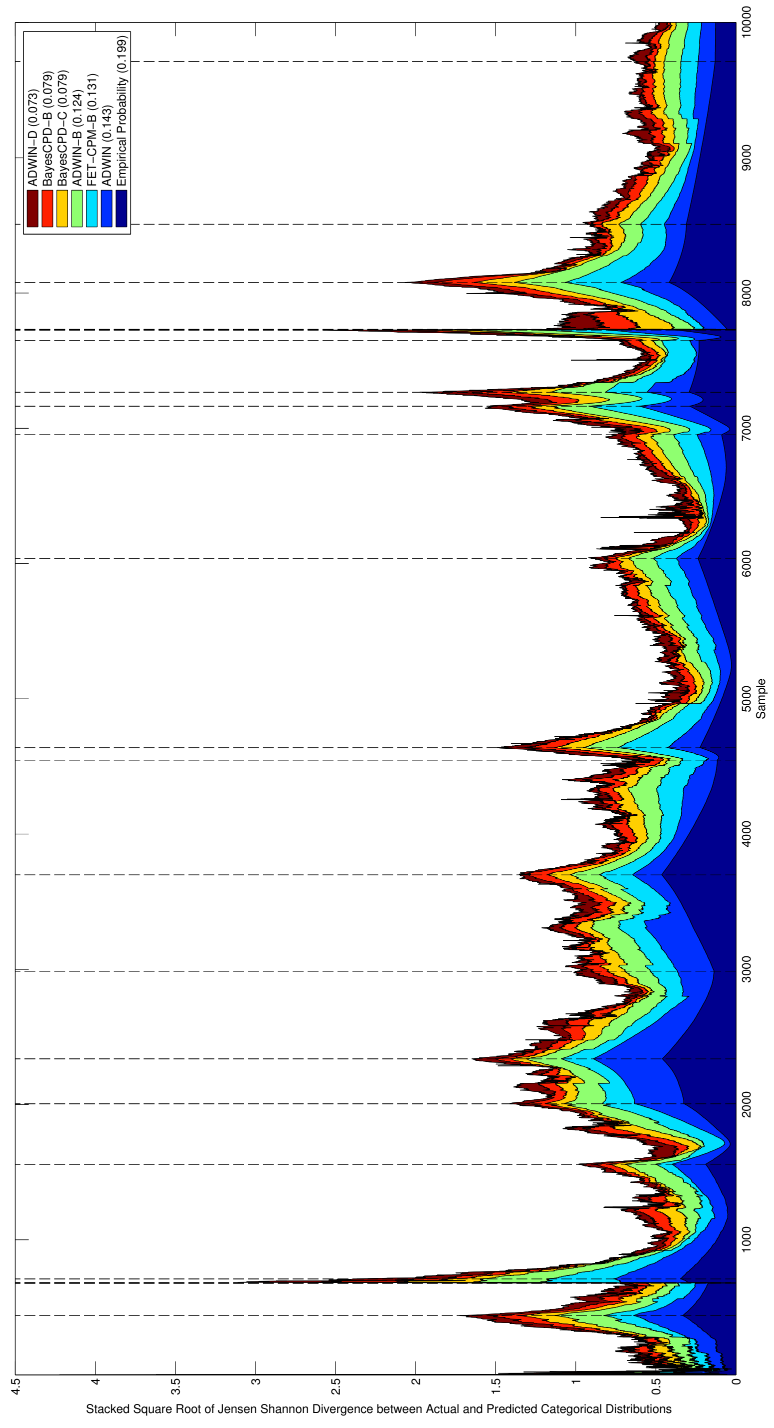
Method Parameters	
ADWIN and ADWIN-B	confidence $\delta = 0.1$, thresholds calculated with Equation 6.2
ADWIN-D	min window size = 10, max window size = 100, threshold $\gamma = 0.3$
BayesCPD-B and BayesCPD-C	change probability $h(0) = 1 \times 10^{-4}$, number of particles = 100
Empirical Probability	N/A
FET-CPM-B	average run length ARL0 = 370, startup = 20, smoothing $\lambda = 0.1$
Changing Categorical Distribution Parameters	
Sudden changes	probability = 2×10^{-3}, duration = 1
Gradual changes 1st type	probability = 2×10^{-3}, duration = time between change points
Gradual changes 2nd type	stepsize $h = 0.05$
Mixed change	equal probability of sudden change, gradual change 1st type, and gradual change 2nd type
Experimental Parameters	
Duration = 1×10^{4}, repeats (when used) = 80	

The first experiment tests sudden changes, the second experiment tests gradual changes of the first type, the third experiment tests gradual changes of the second type, and the fourth and final experiment tests a mixture of all these changes. An example of the results for a single run of each experiment are shown in Figure 6.1 and Table 6.3. The main results, where each experiment is repeated 80 times, are shown in Table 6.4. The main results show that for 5 or fewer categories, ADWIN-D, BayesCPD-B, and BayesCPD-C all have the smallest distances, with small variations between each method. Specifically, ADWIN-D has slightly smaller distances for gradual changes, whereas BayesCPD-B and BayesCPD-C have slightly smaller distances for sudden and mixed changes. For 10 or 20 categories, BayesCPD-C has the smallest distances for all types of changes, and in most cases this is by a relatively large margin. As the number of categories increases, it gets harder to detect changes in individual categories because they are less likely to have a large probability mass shifted to or away from them. The result is that the distances of all the change detection methods increase with the number of categories. One way to mitigate this would be to make each change detection method more sensitive i.e. by increasing δ, $(1 - \gamma)$, $h(0)$, and 1/ARL0. An alternative approach would be to group similar categories together and to model them as one category, although this abstraction may not be possible if the categories are unrelated.



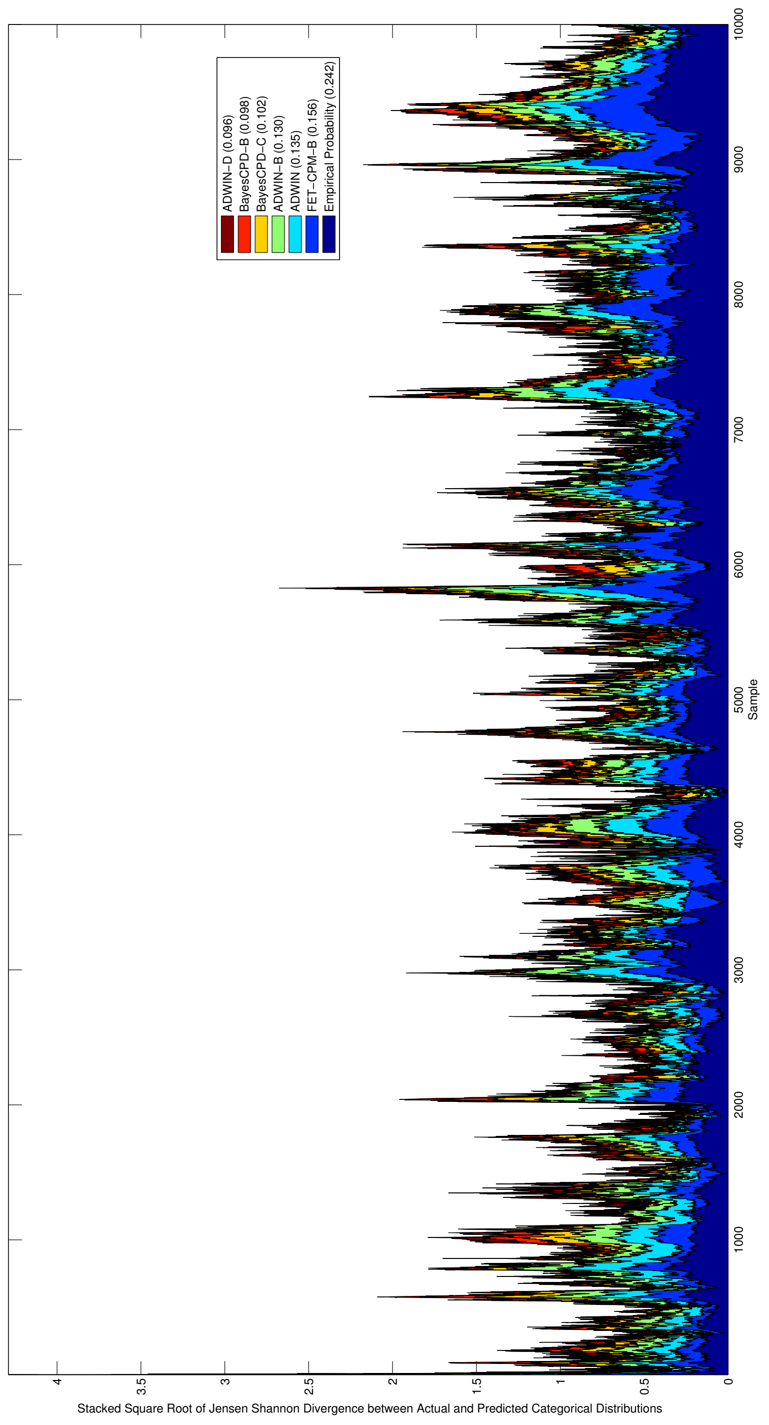
(a) Sudden changes.

Figure 6.1



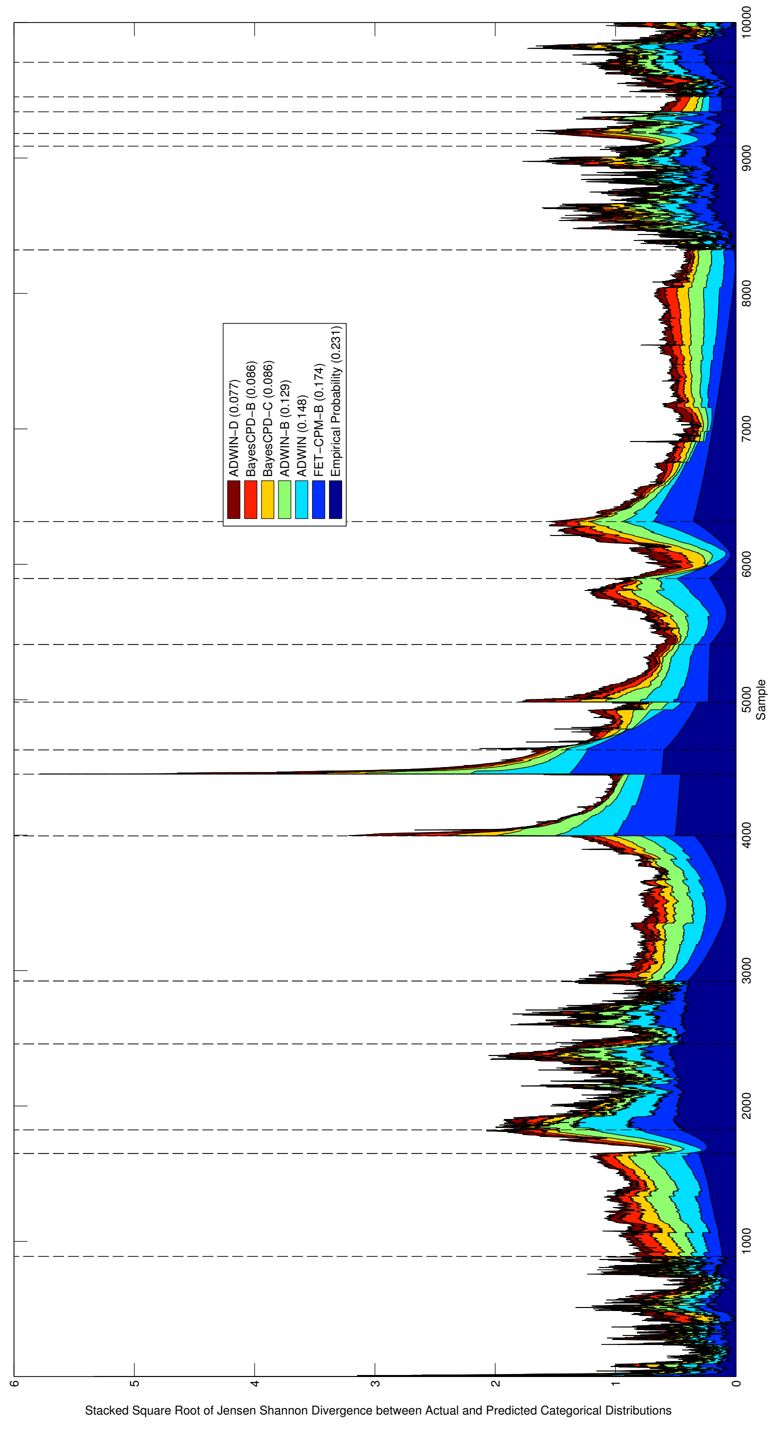
(b) Gradual changes 1.

Figure 6.1



(c) Gradual changes 2.

Figure 6.1



(d) Mixed changes.

Figure 6.1: These stacked area graphs show single run results for change detection experiments. The area of each method represents the distance between its estimated categorical distribution and the actual categorical distribution over time. Distances are measured using the Jensen-Shannon divergence metric. Switches for sudden and gradual changes (first type) occur at vertical dashed lines.

Table 6.3: Example single run change detection results for 3 categories. There is a row for each method. There are columns for sudden changes, gradual changes of the first and second types, and mixed changes. Each entry shows the Jensen-Shannon divergence metric between the method's estimated categorical distribution and the actual categorical distribution averaged over all time.

Method	Sudden Changes	Gradual Changes 1	Gradual Changes 2	Mixed Changes
ADWIN	0.161	0.143	0.135	0.148
ADWIN-B	0.089	0.124	0.130	0.129
ADWIN-D	0.072	**0.073**	**0.096**	**0.077**
BayesCPD-B	0.045	0.079	0.098	0.086
BayesCPD-C	**0.044**	0.079	0.102	0.086
Empirical Probability	0.279	0.199	0.242	0.231
FET-CPM-B	0.175	0.131	0.156	0.174

Table 6.4: Average change detection results for different numbers of categories. There is a row for each method. There are columns for sudden changes, gradual changes of the first and second types, and mixed changes. Each entry shows the Jensen-Shannon divergence metric between the method's estimated categorical distribution and the actual categorical distribution averaged over all time and 80 repeats along with the Standard Error of the Mean (SEM).

(a) 2 categories.

Method	Sudden Changes	Gradual Changes 1	Gradual Changes 2	Mixed Changes
ADWIN	0.078 ± 0.009	0.100 ± 0.005	0.102 ± 0.006	0.092 ± 0.007
ADWIN-B	0.078 ± 0.009	0.100 ± 0.005	0.102 ± 0.006	0.092 ± 0.007
ADWIN-D	0.052 ± 0.006	**0.047 ± 0.002**	**0.063 ± 0.003**	0.054 ± 0.004
BayesCPD-B	**0.035 ± 0.003**	0.053 ± 0.002	0.065 ± 0.003	**0.050 ± 0.003**
BayesCPD-C	0.037 ± 0.004	0.059 ± 0.002	0.069 ± 0.003	0.054 ± 0.003
Empirical Probability	0.21 ± 0.02	0.18 ± 0.02	0.19 ± 0.02	0.19 ± 0.02
FET-CPM-B	0.13 ± 0.02	0.12 ± 0.01	0.12 ± 0.01	0.12 ± 0.02

(b) 3 categories.

Method	Sudden Changes	Gradual Changes 1	Gradual Changes 2	Mixed Changes
ADWIN	0.13 ± 0.02	0.146 ± 0.008	0.146 ± 0.007	0.136 ± 0.009
ADWIN-B	0.10 ± 0.01	0.128 ± 0.005	0.134 ± 0.005	0.116 ± 0.007
ADWIN-D	0.074 ± 0.007	**0.069 ± 0.002**	**0.090 ± 0.003**	0.076 ± 0.004
BayesCPD-B	**0.048 ± 0.004**	0.076 ± 0.003	0.094 ± 0.003	**0.071 ± 0.003**
BayesCPD-C	0.049 ± 0.005	0.080 ± 0.003	0.098 ± 0.003	0.074 ± 0.004
Empirical Probability	0.27 ± 0.02	0.22 ± 0.01	0.26 ± 0.02	0.25 ± 0.01
FET-CPM-B	0.16 ± 0.02	0.145 ± 0.010	0.162 ± 0.009	0.15 ± 0.01

(c) 4 categories.

Method	Sudden Changes	Gradual Changes 1	Gradual Changes 2	Mixed Changes
ADWIN	0.16 ± 0.02	0.169 ± 0.007	0.174 ± 0.006	0.17 ± 0.01
ADWIN-B	0.11 ± 0.01	0.145 ± 0.005	0.157 ± 0.004	0.137 ± 0.008
ADWIN-D	0.092 ± 0.007	**0.085 ± 0.002**	**0.108 ± 0.002**	0.095 ± 0.004
BayesCPD-B	**0.059 ± 0.005**	0.090 ± 0.003	0.115 ± 0.003	**0.088 ± 0.004**
BayesCPD-C	**0.059 ± 0.006**	0.092 ± 0.003	0.117 ± 0.003	0.089 ± 0.004
Empirical Probability	0.29 ± 0.02	0.239 ± 0.010	0.28 ± 0.01	0.27 ± 0.01
FET-CPM-B	0.18 ± 0.02	0.154 ± 0.010	0.180 ± 0.007	0.17 ± 0.01

(d) 5 categories.

Method	Sudden Changes	Gradual Changes 1	Gradual Changes 2	Mixed Changes
ADWIN	0.18 ± 0.02	0.189 ± 0.008	0.194 ± 0.006	0.19 ± 0.01
ADWIN-B	0.13 ± 0.01	0.162 ± 0.005	0.172 ± 0.004	0.151 ± 0.008
ADWIN-D	0.112 ± 0.007	**0.100 ± 0.003**	**0.124 ± 0.002**	0.111 ± 0.004
BayesCPD-B	**0.067 ± 0.006**	0.105 ± 0.003	0.131 ± 0.003	0.099 ± 0.004
BayesCPD-C	**0.067 ± 0.006**	0.103 ± 0.002	0.130 ± 0.003	**0.098 ± 0.004**
Empirical Probability	0.31 ± 0.01	0.258 ± 0.008	0.300 ± 0.009	0.29 ± 0.01
FET-CPM-B	0.19 ± 0.02	0.168 ± 0.009	0.192 ± 0.006	0.18 ± 0.01

(e) 10 categories.

Method	Sudden Changes	Gradual Changes 1	Gradual Changes 2	Mixed Changes
ADWIN	0.24 ± 0.02	0.226 ± 0.007	0.241 ± 0.005	0.23 ± 0.01
ADWIN-B	0.18 ± 0.02	0.193 ± 0.005	0.216 ± 0.003	0.192 ± 0.010
ADWIN-D	0.296 ± 0.005	0.303 ± 0.004	0.296 ± 0.004	0.296 ± 0.005
BayesCPD-B	0.11 ± 0.01	0.156 ± 0.004	0.190 ± 0.003	0.145 ± 0.007
BayesCPD-C	**0.094 ± 0.007**	**0.126 ± 0.002**	**0.167 ± 0.002**	**0.125 ± 0.005**
Empirical Probability	0.34 ± 0.01	0.277 ± 0.006	0.321 ± 0.005	0.322 ± 0.008
FET-CPM-B	0.21 ± 0.02	0.173 ± 0.008	0.211 ± 0.003	0.20 ± 0.01

(f) 20 categories.

Method	Sudden Changes	Gradual Changes 1	Gradual Changes 2	Mixed Changes
ADWIN	0.28 ± 0.02	0.258 ± 0.006	0.284 ± 0.003	0.275 ± 0.010
ADWIN-B	0.23 ± 0.02	0.231 ± 0.005	0.254 ± 0.002	0.235 ± 0.010
ADWIN-D	0.420 ± 0.004	0.436 ± 0.003	0.421 ± 0.003	0.425 ± 0.003
BayesCPD-B	0.355 ± 0.008	0.295 ± 0.004	0.337 ± 0.003	0.333 ± 0.006
BayesCPD-C	**0.125 ± 0.008**	**0.155 ± 0.002**	**0.201 ± 0.001**	**0.156 ± 0.005**
Empirical Probability	0.355 ± 0.008	0.293 ± 0.004	0.336 ± 0.003	0.331 ± 0.006
FET-CPM-B	0.23 ± 0.02	0.194 ± 0.007	0.232 ± 0.002	0.215 ± 0.010

6.3.2 Modelling Reinforcement Learning Agents

There are two experiments in this section. Both experiments use each change detection method and an empirical distribution to play against a variety of popular and state-of-the-art reinforcement learning agents. In each case, this is done by playing a best-response strategy to the prediction of the opponent's distribution (i.e. the predicted reinforcement learning agent's strategy). In the case where the empirical distribution is used, this is simply fictitious play. These reinforcement learning agents include ϵ-greedy Q-Learning, WoLF-PHC, WPL, and PGA-APP.

Initial experiments showed that given the same parameters as in Section 6.3.1, the change detection methods struggle to track the changes in the reinforcement learning agents' strategies because they change more quickly than the categorical distributions in Section 6.3.1. Therefore, these parameters were tuned in this section to make the change detection based agents more competitive by allowing them to detect changes more quickly. Specifically, the confidence, δ, is raised to 7 and the thresholds, ϵ_i, are calculated using the more conservative Equation 6.1 instead of Equation 6.2. Although the threshold, γ, is not changed (i.e. $\gamma = 0.3$), the minimum window size is reduced to 1 from 10. Finally, the constant hazard function, $h(0)$, is raised to $h(0) = 0.12$ for BayesCPD-B, and $h(0) = 0.35$ for BayesCPD-C. For the reinforcement learning algorithms, a standard small fixed exploration rate of 0.05 is used to ensure that they never stop exploring and that their exploration does not significantly affect their rewards. In most situations higher exploration rates would likely decrease the rewards of the change detection methods and the reinforcement learning algorithms. For change detection methods this is because they typically assume a single fixed switching rate for strategy changes, but a high exploration rate effectively introduces a second switching rate. For reinforcement learning algorithms this is because a high exploration rate would make them act uniformly at random more often, which is typically a bad strategy. However, for the games in this section (matching pennies and rock-paper-scissors) a uniform random strategy is a Nash equilibrium strategy and so is not a bad strategy. For the reinforcement learning algorithms their other parameters are tuned to improve rewards. They use a high learning rate of 0.99 and discount factor of 0.99 to learn quickly and to lookahead as far as possible. The step-size is set to 0.05 to learn quickly without constantly jumping outside the strategy space. The losing step-size is set to double the winning step-size as in Bowling and Veloso, 2002 and the prediction length is set to 1 as in Zhang and Lesser, 2010. Finally, the number of repeats is chosen to ensure statistical significance. All of these parameters are summarised in Table 6.5.

Table 6.5: Methods vs reinforcement learning agents parameters.

Method Parameters	
ADWIN and ADWIN-B	confidence $\delta = 7$, thresholds calculated with Equation 6.1
ADWIN-D	min window size = 1, max window size = 100, threshold $\gamma = 0.3$
BayesCPD-B	change probability $h(0) = 0.12$, number of particles = 100
BayesCPD-C	change probability $h(0) = 0.35$, number of particles = 100
Empirical Probability	N/A
FET-CPM-B	average run length ARL0 = 370, startup = 20, smoothing $\lambda = 0.1$
Reinforcement Learning Agents Parameters	
All reinforcement learning agents	exploration rate = 0.05, learning rate = 0.99, discount factor = 0.99
WoLF-PHC, WPL, and PGA-APP	step-size = 0.05
WoLF-PHC	winning step-size = 0.05, losing step-size = 0.1
PGA-APP	prediction length = 1
Experimental Parameters	
Duration = 1×10^4, repeats (when used) = 80	

In the first experiment, the game is iterated matching pennies, and in the second experiment, the game is iterated rock-paper-scissors. For simplicity, each iterated game only has one state, which the agents observe repeatedly. This state is not based on any information such as a memory of interaction. This means that each agent's strategy can be exactly represented by a single categorical distribution, which makes it easier to compare accuracies. The results for iterated matching pennies are shown in tables 6.6, 6.7, and 6.8, and the results for iterated rock-paper-scissors are shown in tables 6.9, 6.10, and 6.11.

The average distances in tables 6.6 and 6.9 show that the ADWIN based methods have the smallest average distances against ϵ-greedy Q-Learning, and that the Bayesian methods have the smallest average distances against WoLF-PHC, WPL and PGA-APP. This implies that the ADWIN based methods are more accurate at modelling agents with deterministic strategies (ignoring exploration), whilst the Bayesian methods are more accurate at modelling agents with mixed strategies. However, the average correct predictions in tables 6.7 and 6.10 show that all modelling methods, except fictitious play and FET-CPM-B, have around the same average correct predictions. This is because, although lower average distances imply higher average accuracies, a model does not need to be completely accurate to produce correct predictions. For example, if an opponent always plays rock in rock-paper-scissors, then a best-response strategy to the distribution that predicts rock with certainty will be the same as the best-response strategy to any distribution where rock has the maximum probability.

All modelling methods, except fictitious play and FET-CPM-B, have high average correct predictions against ϵ-greedy Q-Learning. This is because ϵ-greedy Q-Learning tends to play sequences of the same action, which makes it predictable. The reason it does this is that (ignoring exploration) it deterministically plays the action that it estimates has the highest expected discounted reward, and it takes time to update these estimates. The delay until action a_2 overtakes action a_1 in having the highest expected discounted reward is approximately the amount of time action a_1 will be played continuously.

The average rewards in tables 6.8 and 6.11 show that all modelling agents, except for fictitious play and FET-CPM-B, gain positive average rewards against all reinforcement learning agents. In fact, the modelling agents based on fictitious play and FET-CPM-B have negative average rewards in all cases. Additionally, the average rewards of all modelling agents, except for fictitious play and FET-CPM-B, are generally higher in iterated rock-paper-scissors than in iterated matching pennies. Given that rock-paper-scissors is like a larger version of matching pennies (i.e. it has an extra action), then this implies that the modelling agents may scale better than the reinforcement learning agents to games with more actions. Despite the differences in average distances, the average correct predictions and thus the average rewards remain competitive between the modelling agents (except for fictitious play and FET-CPM-B).

Overall, these results show that by using the parameters in Table 6.5, the ADWIN based change detectors and the Bayesian change detectors can be used to effectively model changing opponent strategies. The reason for this is that best-response strategies against their models are able to produce positive average rewards per game, and outperform the reinforcement learning agents.

Table 6.6: Matching pennies average distances.

	ϵ-greedy Q-Learning	WoLF-PHC	WPL	PGA-APP
ADWIN	**0.224**	0.529	0.544±0.0003	0.525
ADWIN-B	0.225	0.529	0.551	0.524
ADWIN-D	**0.224**	0.529	0.544±0.0003	0.525
BayesCPD-B	0.386	0.136	**0.116**	0.145
BayesCPD-C	0.396	**0.13**	0.117	**0.138**
Fictitious Play	0.496	0.234±0.001	0.149	0.342
FET-CPM-B	0.5	0.269±0.001	0.224±0.0002	0.382

Table 6.7: Matching pennies average correct predictions.

	ϵ-greedy Q-Learning	WoLF-PHC	WPL	PGA-APP
ADWIN	**0.849**	**0.539**	0.506	0.543
ADWIN-B	0.847	0.537	0.508	0.544
ADWIN-D	**0.849**	**0.539**	0.506	0.543
BayesCPD-B	0.847	0.536	0.508	**0.55**
BayesCPD-C	0.848	0.538	**0.509**	0.543
Fictitious Play	0.448	0.46	0.46	0.471
FET-CPM-B	0.431±0.0002	0.39±0.0002	0.386±0.0002	0.36±0.0002

Table 6.8: Matching pennies average rewards.

	ϵ-greedy Q-Learning	WoLF-PHC	WPL	PGA-APP
ADWIN	**0.698±0.0001**	**0.079±0.0001**	0.012±0.0002	0.086±0.0001
ADWIN-B	0.694	0.074±0.0001	0.016±0.0001	0.088±0.0001
ADWIN-D	**0.698±0.0001**	**0.079±0.0001**	0.012±0.0002	0.086±0.0001
BayesCPD-B	0.694±0.0001	0.072±0.0001	0.017±0.0001	**0.1±0.0001**
BayesCPD-C	0.697±0.0001	0.076±0.0001	**0.019±0.0002**	0.087±0.0001
Fictitious Play	-0.104	-0.08	-0.079	-0.058
FET-CPM-B	-0.138±0.0003	-0.219±0.0003	-0.228±0.0004	-0.279±0.0003

Table 6.9: Rock-paper-scissors average distances.

	ϵ-greedy Q-Learning	WoLF-PHC	WPL	PGA-APP
ADWIN	**0.325**	0.593	0.656	0.584
ADWIN-B	**0.325**	0.592	0.656	0.585
ADWIN-D	**0.325**	0.593	0.656	0.584
BayesCPD-B	0.467	0.219	0.167	**0.224**
BayesCPD-C	0.494	**0.215**	**0.15**	**0.224**
Fictitious Play	0.588	0.405	0.296	0.424
FET-CPM-B	0.592	0.471	0.36±0.0001	0.466

Table 6.10: Rock-paper-scissors average correct predictions.

	ϵ-greedy Q-Learning	WoLF-PHC	WPL	PGA-APP
ADWIN	0.768	**0.458**	0.368	**0.468**
ADWIN-B	0.769	**0.458**	**0.369**	**0.468**
ADWIN-D	0.768	**0.458**	0.368	**0.468**
BayesCPD-B	**0.78**	0.446	0.365	0.466
BayesCPD-C	0.636±0.0001	0.439	0.364	0.456
Fictitious Play	0.306	0.313	0.308	0.31
FET-CPM-B	0.221±0.0001	0.19±0.0001	0.21±0.0001	0.2±0.0001

Table 6.11: Rock-paper-scissors average rewards.

	ϵ-greedy Q-Learning	WoLF-PHC	WPL	PGA-APP
ADWIN	**0.658±0.0002**	0.186±0.0001	0.051±0.0001	0.2±0.0001
ADWIN-B	0.657±0.0001	0.185±0.0001	0.053±0.0001	0.199±0.0001
ADWIN-D	**0.658±0.0002**	0.186±0.0001	0.051±0.0001	0.2±0.0001
BayesCPD-B	0.587±0.0002	**0.207±0.0001**	0.055±0.0001	0.232±0.0001
BayesCPD-C	0.584±0.0002	0.204±0.0001	**0.057±0.0001**	**0.233±0.0001**
Fictitious Play	-0.074	-0.045	-0.041	-0.041
FET-CPM-B	-0.259±0.0002	-0.33±0.0003	-0.245±0.0003	-0.291±0.0002

6.4 Combining Sequence Prediction and Change Detection

In previous chapters, particularly Chapter 4, it has been shown how sequence prediction can be used to predict a probability distribution over the future actions of an opponent based on a sequence of observations from the past. These observations could be composed of, for example, previous states, or actions, or both.

The predictions can be seen as conditional probability distributions over the opponent's actions, where the conditions are subsequences of observed sequences. Each conditional distribution is assumed to be stationary, but this may be an incorrect assumption. For example, it could be the case that in the prisoner's dilemma there is an opponent who starts off using the tit-for-tat strategy, but eventually switches to the always defect or the always cooperate strategies depending on how much it trusts its opponent. Some sequence prediction methods have ways to handle changing distributions. For example, ELPH will prune a conditional distribution if it has a high entropy because this implies that it is close to a uniform random distribution and thus is not very useful for predicting. Change detection methods offer an alternative and more proactive approach for handling changing distributions by detecting their change points and discarding irrelevant samples prior to those change points. Thus, sequence prediction methods could use change detection methods to model changing conditional distributions.

6.5 Chapter Summary

This chapter takes another look at the problem of modelling an opponent's strategy (i.e. categorical distributions) when they are changing. These changes may be due to, for example, the opponent learning. In order to accurately model a categorical distribution, it is important to only use samples that actually originate from it. This means that if and when the categorical distribution changes, then samples that do not represent it should be ignored or forgotten. It was discussed how this could be done by using an adaptive window of past observations such that the window would be shortened as the distribution changed to remove unrepresentative samples. In order to detect these changes, variations of three state-of-the-art change detection methods were considered, which include ADaptive WINdowing (ADWIN) by Bifet and Gavaldà, 2007, Bayesian Change Point Detection (BayesCPD) by Adams and MacKay, 2007 as well as Fearnhead and Liu, 2007, and Fisher's Exact Test Change Point Model (FET-CPM) by Ross, Tasoulis, and Adams, 2013. Some of these variations were first proposed in this chapter to handle categorical distributions. The main contribution of this chapter is to experimentally compare these methods against each other and against an empirical distribution at predicting changing categorical distributions.

In the first comparison, the categorical distributions were changing either suddenly, or gradually, or a mixture of both. The results show that for small numbers of categories (2-5), the BayesCPD methods maintain the most accurate models for sudden and mixed changes, whilst ADWIN-D maintains the most accurate models for gradual changes. However, for large numbers of categories (10 or 20), BayesCPD-C maintains the most accurate models for all types of changes. Also, as the number of categories is increased, all methods perform worse at different degrees of degradation due to each category generally having less probability mass making probability mass shifts between categories harder to detect. In the second comparison, the categorical distributions were actually the strategies of popular and state-of-the-art reinforcement learning agents, which were changing according to their update rules and feedback. In other words, the change detection methods (and the empirical distribution) were used to predict the strategies of the reinforcement learning agents, which in turn were used to calculate and play best-response strategies in iterated matching pennies and iterated rock-paper-scissors. The results here show that the ADWIN methods and the BayesCPD methods can effectively model changing opponent strategies such that best-response strategies against their models gain positive average rewards against the reinforcement learning agents. Although in general, the BayesCPD methods are the most accurate against the reinforcement learning agents. Finally, it was discussed how these methods could improve sequence prediction methods, which could be a direction for future research.

Chapter 7

Dynamic Opponent Modelling Self-Play Convergence

Ideally, we want each agent in a multi-agent system to consistently learn and change its strategy to increase its expected rewards. If an agent could maintain this behaviour, then eventually it would learn and converge to a best-response strategy that maximises its expected rewards against the other agents' strategies. If all agents could maintain this behaviour, then eventually they would learn and converge to a Nash equilibrium. However, it is difficult to devise learning rules that will consistently change an agent's strategy to increase its expected rewards, especially since other agents' strategies can affect those rewards. Crawford's puzzle illustrates this (see Section 3.2.3), where seemingly natural learning rules did not result in convergence to mixed strategy Nash equilibria despite good conditions (i.e. simple problems, clear and relevant feedback, unique equilibria, etc.). This is why a large part of the literature about learning in multi-agent systems focuses on finding learning rules that, when at least used in self-play, will result in agents' strategies converging to a Nash equilibrium[1]. This is especially true for mixed strategy Nash equilibria, which tend to be more difficult to converge to than pure strategy Nash equilibria. The idea behind using opponent modelling to help convergence to Nash equilibria is that if an agent knows its opponents' strategies, and the rules of the game (including its own rewards), then it can directly calculate a best-response strategy.

In this chapter, which is based on [Mealing and Shapiro, 2015], the goal is to

[1]Abdallah and Lesser, 2008; Awheda and Schwartz, 2013; Banerjee and Peng, 2003; Bowling, 2005; Bowling and Veloso, 2002; Butterworth and Shapiro, 2009; Zhang and Lesser, 2010.

address the question, will convergence be enhanced if each agent assumes that the other agents are changing their strategies over time. This will be studied using simultaneous-move games with two or three actions, and two or three players. A comparison is drawn between fictitious play, which assumes that the opponent uses a stationary strategy, with two new variants that remove this assumption and explicitly assume that the opponent uses a dynamic strategy. The opponent's strategy is predicted using a sequence prediction method in the first variant, and a change detection method in the second variant. At each step each method observes the opponent's action, predicts its strategy, and then plays a best-response strategy to the predicted strategy.

Specifically the comparison looks at the convergence in self-play of these variants and fictitious play to mixed strategy Nash equilibria. Only the empirical distributions of plays over games are considered because they almost always play pure strategies. Experiments find that these variants converge faster than fictitious play. However, unlike in fictitious play, these variants do not always exactly converge to the Nash equilibria. For change detection, this is a very small number of cases, but for sequence prediction there are many. Combining each variant with fictitious play improves its convergence, reducing these cases. Also, unlike fictitious play, these variants converge to the mixed strategy Nash equilibria in Shapley's and Jordan's games, which are considered difficult [Leslie, 2003].

7.1 Fictitious Play Example

Fictitious play and its convergence properties are described in Section 3.2.10, from which there are three key points about fictitious players in self-play:

1. They cannot converge to a mixed strategy Nash equilibrium because each one always plays a pure strategy.
2. Their empirical distributions of plays can converge to a mixed strategy Nash equilibrium in certain games.
3. If their empirical distributions of plays converge to a Nash equilibrium and they are each independent from one another, then their expected rewards will also converge to those at that Nash equilibrium.

This last point captures the type of convergence that this chapter investigates.

In this example, two fictitious players are played against each other in self-play in an iterated game of matching pennies. The row player wins if it matches the action of the column player; otherwise the column player wins. A win (loss) is worth $(-)1$. This is shown in Table 3.5. Let heads be represented by 1, tails be represented by -1, x be the mean of the row player's actions, and y be the mean of the column player's actions. So, for example, $x = 0$ would mean that the row player has played equal numbers of heads and tails. The row player updates y after observing each of the column player's actions, and will play its sign. Whilst the column player updates x after observing each of the row player's actions, and will play *minus* its sign. The expected dynamics of x and y are

$$x(t) = \left(1 - \frac{1}{t}\right) x(t-1) + \left(\frac{1}{t}\right) \mathrm{sign}(y(t-1)), \tag{7.1}$$

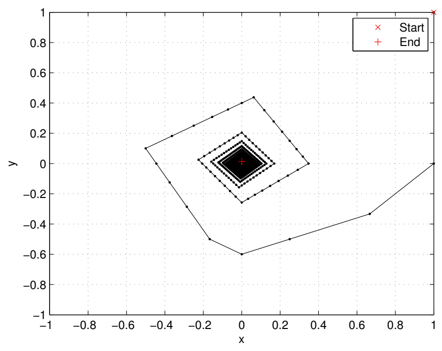
$$y(t) = \left(1 - \frac{1}{t}\right) y(t-1) - \left(\frac{1}{t}\right) \mathrm{sign}(x(t-1)), \tag{7.2}$$

$$\mathrm{sign}(z) = \begin{cases} 1 & \text{if } z > 0, \\ 0 & \text{if } z = 0, \\ -1 & \text{if } z < 0. \end{cases}$$

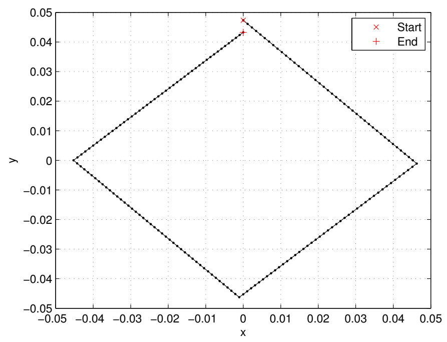
If a player has played equal numbers of heads and tails, then under fictitious play dynamics, its opponent would play a (usually uniform) random action. Thus, the expectation of its opponent's play in this situation is zero. This is why these are the expected dynamics. The players' actions will never converge, because the signs of x and y will never stop changing. However, the means *do* converge, albeit slowly. To illustrate these expected dynamics, 10000 iterations of recurrence relations 7.1 and 7.2 were ran. The initial values of x and y were each randomly set to either -1 or 1 with equal probability. The results are shown in figures 7.1a, 7.1b, 7.1c, and 7.1d. Note that many two-player, two-action, zero-sum, normal-form games with mixed strategy Nash equilibria will be similar, with x and y measuring the difference between the strategy profile and the equilibrium.

Viewing this as a dynamical system, the following can be said about equations 7.1 and 7.2:

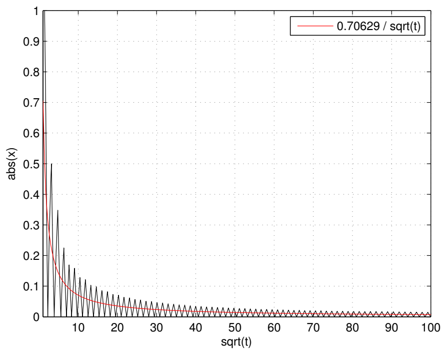
1. The point $(x = 0, y = 0)$ is a fixed point.
2. The system cycles towards the origin, by switching strategies, as seen in figures 7.1a and 7.1b.



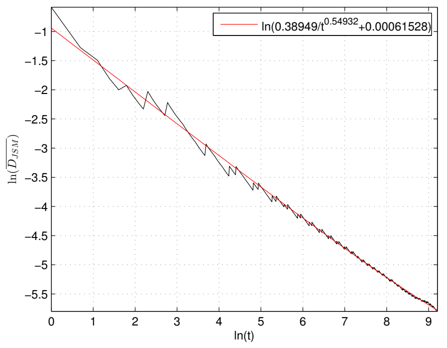
(a) At the start, $x(1) = 1$, $y(1) = 1$, and at the end, $x(10000) = 0.000080$, $y(10000) = 0.014$. They are converging to the Nash equilibrium at the centre, but more slowly as time goes on as the distance between points is decreasing.



(b) One cycle between iterations 866 and 1040. At the start, $x(866) \approx 0$ but positive, $y(866) = 0.047$, and at the end, $x(1040) \approx 0$ but positive, $y(1040) = 0.043$. Note that at iterations 866 and 1040, x has just switched signs i.e. at iterations 865 and 1039 $x \approx 0$ but negative.



(c) The abs(x) is converging towards 0.



(d) The average Jensen-Shannon divergence metric is converging towards 0. See Section 7.4.1 for a definition of the average Jensen-Shannon divergence metric.

Figure 7.1: Expected dynamics of fictitious play in self-play in matching pennies over 1×10^4 iterations. The parameters of the best-fit lines were calculated using MATLAB's Trust-Region-Reflective Least Squares algorithm with Bisquare weights *Least-Squares Algorithms*.

3. The period of a cycle grows *at least* linearly with time.
4. The amount the system moves towards the origin per cycle decreases *at least* inversely with time.
5. As a consequence of 3 and 4, the convergence rate is $\Theta(1/\sqrt{t})$, where t is time (iteration).

Point 1 is obvious, Appendix D shows points 2, 3, 4, and 5.

The fact that fictitious play undergoes empirical Nash convergence here, but very slowly, as shown empirically in figures 7.1a, 7.1b, 7.1c, and 7.1d, as well as theoretically in Appendix D, is part of the motivation of this chapter. Convergence is very slow because each agent takes an increasingly long time to change its strategy in response to the change in its opponent's strategy. If an agent could identify its opponent's strategy switches more quickly, then it might converge faster, perhaps optimally like $1/t$, as well as in more situations. This is the idea that is investigated in this chapter.

7.2 Extensions to Fictitious Play

In this chapter, only the traditional fictitious play algorithm as described in Section 3.2.10 is considered. However, to help put the work in this chapter into context, some extensions to it are briefly described. These extensions can allow fictitious play to model changing opponent strategies, to use mixed strategies, and can improve its convergence to solution concepts. Two popular extensions, introduced by Fudenberg and Levine, 1998, are geometric fictitious play and stochastic fictitious play. Geometric fictitious play can model changing opponent strategies. It works by giving bigger weights to more recent opponent actions when updating opponent action probabilities. In comparison to the traditional update shown in Equation 3.45, the only change is that the factor $1/t$, where t is the iteration, is replaced by a constant $z \in [0, 1]$. The constant z is a "forgetting factor", with higher values placing less weight on past opponent actions. Stochastic fictitious play can play mixed strategies and also introduces exploration. It does this by smoothing the best-response function, which means that instead of selecting the strategy with the maximum expected reward, strategies are selected with probabilities that are proportional to their expected rewards. A common approach to this is for player i to play strategy σ_i with probability

$$\Pr(\sigma_i) = \frac{e^{\overline{u}_i(\sigma_i, \sigma_{-i})\lambda^{-1}}}{\sum_{\sigma_i'} e^{\overline{u}_i(\sigma_i', \sigma_{-i})\lambda^{-1}}}, \tag{7.3}$$

where λ is a randomisation parameter. As λ approaches zero, this smoothed best-response becomes equivalent to a regular best-response. Various extensions are examined by Ny, 2006, who looks at traditional (discrete-time) fictitious play, stochastic (smooth) fictitious play, continuous-time fictitious play, and dynamic fictitious play. Two more extensions are proposed by Smyrnakis and Leslie, 2010, 2011, which can model a changing opponent strategy based on recent observations. The first uses a particle filter algorithm, whilst the second uses a heuristic rule to adaptively update the weights of opponent actions.

7.3 Sequence Prediction and Change Detection Self-Play Convergence

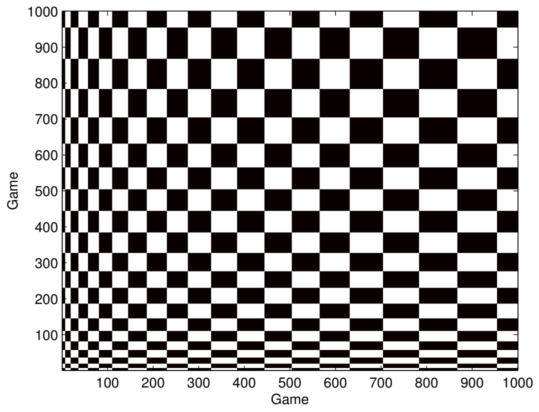
The main question in this chapter is: will convergence in self-play be enhanced if each agent assumes that the other agents are changing their strategies over time? There are two aspects of convergence that can be enhanced, firstly its speed, and secondly its applicability. Experiments in this chapter will look at both aspects, firstly by modelling convergence speed, and secondly by looking at two games where the empirical distributions of plays of fictitious players in self-play do not converge to mixed strategy Nash equilibria, namely Shapley's game and Jordan's game. The advantage shared by sequence prediction and change detection methods over fictitious play is that they do not rely on the typically erroneous assumption that the opponent is using a stationary strategy. By avoiding this assumption, these methods can potentially maintain more accurate opponent models. A more accurate opponent model could allow for more accurate best-response strategies to be calculated, enhancing an agent's ability to act rationally and possibly learn a Nash equilibrium in self-play. Thus, to test this, this chapter compares fictitious play with two new variants. The opponent's strategy is predicted by the first variant using a sequence prediction method, and by the second variant using a change detection method.

Initial experiments found that the empirical distributions of plays of some sequence prediction methods and some change detection methods do not always converge to Nash equilibria in self-play in two-player, two-action, normal-form games derived from generalised matching pennies. The result is sometimes that whilst one agent's empirical distribution of plays would converge to its portion of the Nash equilibrium, the other player's empirical distribution of plays would converge some distance away from its portion of the Nash equilibrium. One possibility to improve the convergence of these methods is to combine each of them with fictitious play in such a way as to keep the convergence properties of fictitious play, whilst taking advantage of the ability of the sequence prediction and change detection methods to model changing strategies. This chapter tests this through two hybrid methods, the first combines sequence prediction with fictitious play, and the second combines change detection with fictitious play.

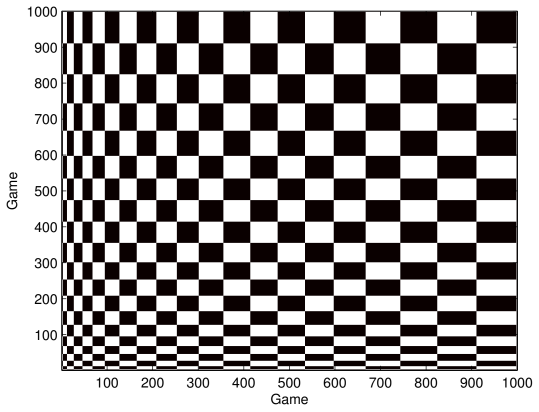
Fictitious play assumes that an opponent's action probability is the ratio of the number of times that it has played that action to the total number of actions that it has played. Thus, each opponent action observation has less power to change fictitious play's modelled opponent action probabilities than any prior opponent action observations. For example, in iterated matching pennies, if fictitious play observes the opponent play heads for the first n games, then the opponent will have to play tails for at least n more games before fictitious play changes its estimated best-response strategy. As more games are played, it becomes harder and harder to change fictitious play's modelled opponent action probabilities, and usually its estimated best-response strategy because it weights all observations equally. An exception to this is if an opponent plays actions approximately an equal number of times. In this situation, future observations still have less of an effect on action probabilities, but the estimated best-response strategy could quickly change. For example, if in rock-paper-scissors the opponent plays rock, then paper, then scissors in a loop, then fictitious play's estimated opponent action probabilities would converge to 1/3 and become harder and harder to change. However, the estimated best-response strategy would go from rock or paper or scissors, then to paper, then to paper or scissors also in a loop.

Sequence prediction and change detection methods can avoid being locked into playing increasingly long sequences of the same best-response strategy. This is because they can, in general, recognise when the same strategy is used for many games. For example, in iterated matching pennies, imagine an opponent who plays the sequence $(H, T, H, H, T, T, T, H, H, H, \dots)$. A sequence prediction method would model $\Pr(H|H)$ and $\Pr(T|T)$, and these probabilities would approach one as the number of games increased. A change detection method may initially model each action as having equal probability. However, eventually it would assume a change has occurred and reset the model, after which it would only observe a pure strategy for a number of games. When the opponent eventually switches to its other pure strategy and plays it for a number of games, this would stand out clearly as a change point.

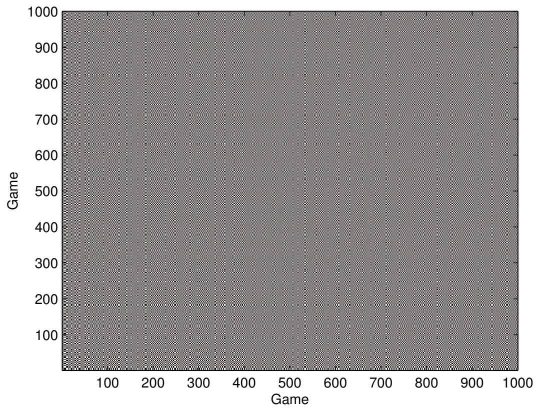
To illustrate fictitious play playing longer and longer sequences of the same action in self-play, and the other methods avoiding this behaviour, each method was played in self-play in matching pennies. Figure 7.2 shows the results i.e. recurrence plots for each method in self-play in matching pennies where FP is fictitious play, SP is sequence prediction, SPFP is sequence prediction and fictitious play, CD is change detection, and CDFP is change detection and fictitious play. The x and y axes represent games, and each point is either black if the action between both games is the same, or white otherwise. In each plot, there is a checkerboard structure, which is a result of the player oscillating between playing heads and tails. For fictitious play, the size of the rectangles increases with the number of games, which shows that each fictitious player is playing longer and longer sequences of the same action. Whereas for the other methods, the rectangles do not always increase in size with the number of games. For sequence prediction, change detection, and change detection with fictitious play, the size of the rectangles remain relatively small compared to fictitious play. For sequence prediction with fictitious play, it looks as if the plots for fictitious play are repeating. This is because the method is mostly acting as fictitious play would, up until the point where sequence prediction momentarily takes over the predictions. In comparison to fictitious play, each other method reduces the temporal correlation between a player's actions, which has two advantages. Firstly, it makes the player less predictable, and secondly, the player is more likely to get the expected reward of its empirical distribution of plays.



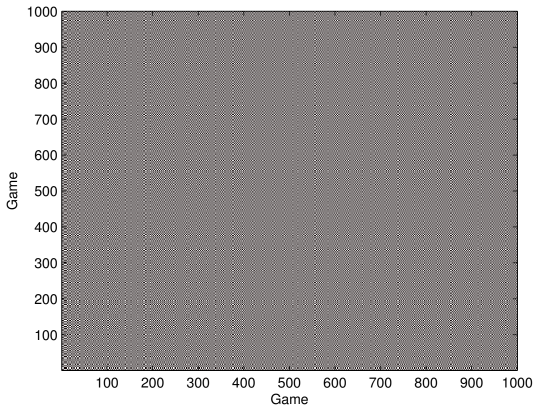
(a) FP player 1.



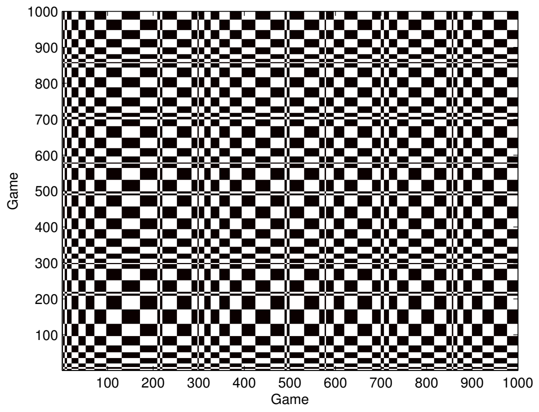
(b) FP player 2.



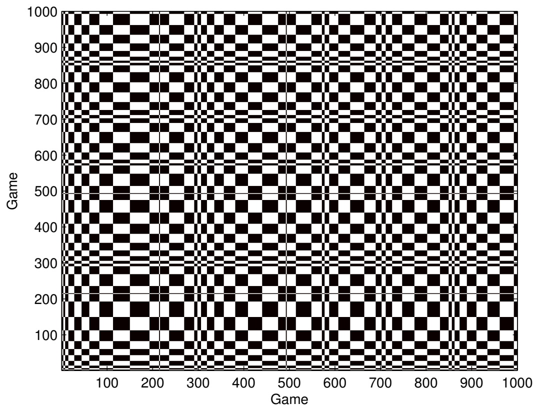
(c) SP player 1.



(d) SP player 2.

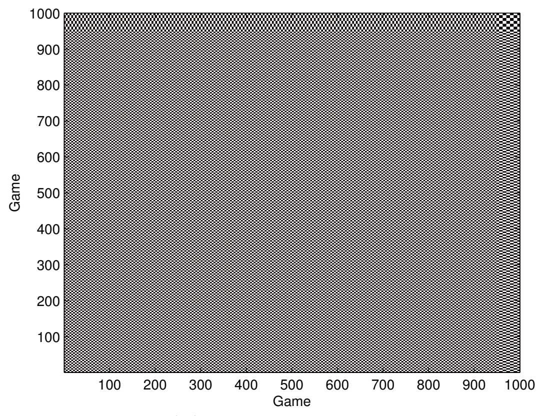


(e) SPFP player 1.

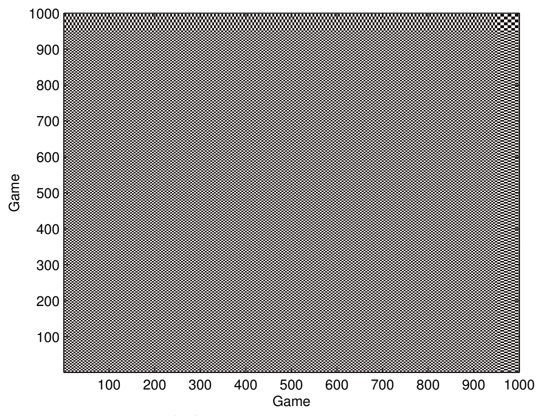


(f) SPFP player 2.

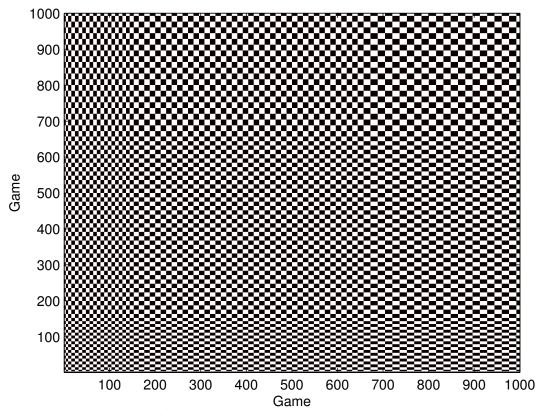
Figure 7.2



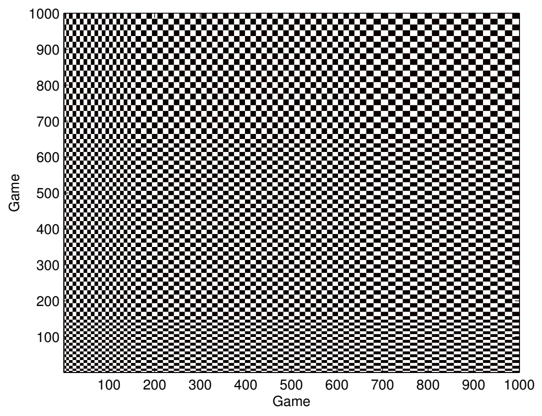
(g) CD player 1.



(h) CD player 2.



(i) CDFP player 1.



(j) CDFP player 2.

Figure 7.2: Recurrence plots for each method in self-play in matching pennies where FP is fictitious play, SP is sequence prediction, SPFP is sequence prediction and fictitious play, CD is change detection, and CDFP is change detection and fictitious play. The x and y axes represent games, and each point is either black if the action between both games is the same, or white otherwise.

7.3.1 Combining Sequence Prediction and Fictitious Play

This method plays a best-response strategy to the opponent's empirical distribution of plays, just as fictitious play would, unless the sequence prediction method predicts an opponent action will occur with a probability greater than a threshold θ. In this case, it plays a best-response strategy to the opponent strategy predicted by the sequence prediction method. This method is shown in Algorithm 16.

Algorithm 16 Sequence Prediction and Fictitious Play

```
Require: Fictitious player $f$, sequence predictor $s$, threshold $0 \leq \theta \leq 1$.
 1: use $f$ to observe $a^t_{\text{opp}}$ (an opponent action at time $t$).
 2: use $s$ to observe $a^t_{\text{opp}}$.
 3: function GET_ACTION
 4:     use $s$ to predict opponent action categorical distribution $D_s$.
 5:     for each opponent action $a_{\text{opp}}$ in $D_s$ do
 6:         if action probability in $D_s$ is above threshold $D_s(a_{\text{opp}}) \geq \theta$ then
 7:             return best-response to $D_s$,
 8:         end if
 9:     end for
10:     use $f$ to predict opponent action categorical distribution $D_f$.
11:     return best-response to $D_f$.
12: end function
```

7.3.2 Combining Change Detection and Fictitious Play

This method attempts to detect changes in and models the opponent's strategy using the Bayesian online change detection method proposed by Fearnhead and Liu, 2007 as well as by Adams and MacKay, 2007. This method also uses the particle filter suggested by Fearnhead and Liu, 2007 to place an upper limit on the number of possible runlengths, and in turn, the memory and time requirements. In Chapter 6 this method was called BayesCPD and experiments were performed with two different versions. The first version, BayesCPD-B, models each category as a Bernoulli distribution, and runs a separate instance of the change detection method on each distribution. The second version, BayesCPD-C, uses a categorical model, and runs one instance of the change detection method on the whole distribution. This method modifies the probabilities returned by a model. It does this by interpolating between the model's probabilities and the empirical probabilities such that as the runlength increases, the weight of the model's probabilities increases. This implementation is shown in Algorithm 17.

Algorithm 17 Bayes Model Modified Probability

```
Require: A list of runlengths $r \leftarrow (r_1, r_2, \dots, r_n)$, a list of runlength weights
    $w \leftarrow (w_1, w_2, \dots, w_n)$, a list of the model's probabilities for each runlength
    $\mu \leftarrow (\mu_1, \mu_2, \dots, \mu_n)$, the empirical probabilities $\mu_t$, and a weighting factor
    $f \in \mathbb{R}$.
1: Initialise a list of modified probabilities for each runlength $\mu' \leftarrow ()$.
2: for $i \leftarrow 1$ to $n$ do
3:     Calculate the modified probabilities for runlength $r_i$, $\mu'_i \leftarrow \frac{r_i\mu_i + f\mu_t}{r_i + f}$.
4: end for
5: Initialise overall probabilities $M \leftarrow 0$.
6: for $i \leftarrow 1$ to $n$ do
7:     Calculate overall probabilities using the new runlength probabilities $M \leftarrow$
    $M + \mu'_i w_i$.
8: end for
9: return $M$
```

7.4 Empirical Results for Self-Play Convergence

The following experiments compare the convergence in self-play of fictitious play to variants of it. Specifically, since each of the algorithms play pure strategies, the comparison is between the convergence of their empirical distributions of plays. For each game, the distances of their empirical distributions of plays from the unique mixed strategy Nash equilibrium are measured. Distances are measured using the Jensen-Shannon divergence metric as defined in Section 7.4.1. From these distances, estimates of their convergence speeds are calculated. The notion of empirical distributions of plays converging to a Nash equilibrium is a very weak form of convergence, which will be called *empirical Nash convergence.* Formally, an empirical distribution, and its convergence to another distribution, is defined as follows. Given a finite set, $\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_k\}$, and an infinite sequence of elements from $\mathcal{A}$, $S = (\alpha^1, \alpha^2, \dots)$, $\alpha^j \in \mathcal{A}$, the empirical distribution of S at time t is

$$P_S^t(\alpha_i \in \mathcal{A}) = \frac{1}{t}\sum_{j=1}^{t} [\![\alpha_i = \alpha^j]\!] \tag{7.4}$$

$$= \left(1 - \frac{1}{t}\right) P_S^{t-1}(\alpha_i \in \mathcal{A}) + \left(\frac{1}{t}\right) [\![\alpha_i = \alpha^t]\!], \tag{7.5}$$

where $[\![\cdot]\!]$ is the Iverson bracket such that $[\![\phi]\!] = 1$ if the predicate ϕ is true, otherwise $[\![\phi]\!] = 0$. Given a probability distribution over $\mathcal{A}$, $P(\alpha_i \in \mathcal{A})$,

Definition The empirical distribution of S converges to P if for any $\epsilon > 0$, and for any divergence measure $D(\cdot||\cdot)$ between distributions, there exists a time t_ϵ such that $D(P_S^t||P) < \epsilon$ for all times $t > t_\epsilon$.

Thus, given a Nash equilibrium, empirical Nash convergence is defined as each player's empirical distribution of plays converging to their strategy in this Nash equilibrium.

The first experiment takes another look at matching pennies. The second experiment looks at various two-player, two-action, normal-form games derived from generalised matching pennies. The third experiment looks at Shapley's game. Finally, the fourth experiment looks at Jordan's game. In all of the experiments, the sequence prediction method is Entropy Learned Pruned Hypothesis Space (ELPH) by Jensen et al., 2005 with a short-term memory size of $k = 1$ and an entropy threshold of $H_l = 1$, whilst the change detection method is Bayesian Change Point Detection using Categorical distributions (BayesCPD-C) with a switching rate of 1×10^{-4} and 100 particles.

In each experiment, the hybrid algorithms, which try to improve the players' empirical Nash convergence, are also tested. For sequence prediction and fictitious play, the threshold, θ, is set to $\theta = 0.95$. For change detection and fictitious play the factor, f, is set to $f = 750$.

7.4.1 Measuring Nash Equilibrium Convergence

To measure the convergence to a Nash equilibrium, we need to be able to measure the difference between each player's strategy and its Nash equilibrium strategy. In a normal-form game, each of these strategies can be represented as a discrete probability distribution. Thus, we want to be able to measure the difference between two discrete probability distributions P and Q. To do this, the Jensen-Shannon divergence metric is used, which is calculated using the Jensen-Shannon divergence, and is based on the Kullback-Leibler divergence. In particular, for each player, the Jensen-Shannon divergence metric between its empirical distribution of plays and its Nash equilibrium strategy is calculated, and the average of these values for each player are taken to give an average Jensen-Shannon divergence metric i.e.

$$\overline{D_{JSM}} = \frac{1}{n}\sum_{i=1}^{n} D_{JSM}(\sigma_i || \sigma_i^*) \tag{7.6}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \sqrt{\frac{1}{2}\left[\left\{\sum_j \sigma_i(j)\log_2\left(\frac{\sigma_i(j)}{M_i(j)}\right)\right\} + \left\{\sum_j \sigma_i^*(j)\log_2\left(\frac{\sigma_i^*(j)}{M_i(j)}\right)\right\}\right]}, \tag{7.7}$$

$$\text{where } M_i(j) = \frac{\sigma_i(j) + \sigma_i^*(j)}{2},$$

where n is the number of players, σ_i is player i's empirical distribution of plays, and σ_i^* is player i's Nash equilibrium strategy. Here it is assumed that σ_i and σ_i^* can each be represented by a discrete probability distribution, which is the case for a mixed strategy in a normal-form game.

7.4.2 Maximum Convergence Rate

This section puts forward a maximum convergence rate of the average Jensen-Shannon divergence metric, which is used to measure Nash equilibrium convergence, by relating it to the maximum convergence rate of an empirical probability. Specifically, it is argued that the maximum convergence rate of an empirical probability is like $O(1/i)$, where i is the iteration number, and that this must also be the maximum convergence rate of the average Jensen-Shannon divergence metric.

Empirical Probability

Given a sequence of events,

$$S_0 = (a_1, a_2, \ldots, a_n), \tag{7.8}$$

where $a_i \in A$ for $1 \leq i \leq n$ and A is the set of all possible events. The sequence of empirical probabilities for an event $\alpha \in A$ is

$$S_1 = \left(\frac{\sum_{i=1}^{1} [\![a_i = \alpha]\!]}{1}, \frac{\sum_{i=1}^{2} [\![a_i = \alpha]\!]}{2}, \ldots, \frac{\sum_{i=1}^{n} [\![a_i = \alpha]\!]}{n} \right), \tag{7.9}$$

where $[\![\cdot]\!]$ is the Iverson bracket such that $[\![\phi]\!] = 1$ if the predicate ϕ is true, otherwise $[\![\phi]\!] = 0$.

We want to know how quickly S_1 can converge to probability, $0 \leq p^* \leq 1$, if it does so as quickly as possible. To do this we can choose S_0 to be any sequence of events, where each event is in A. To make S_1 converge to p^* as quickly as possible we set each term in S_1 as close to p^* as possible giving

$$S_2 = \left(\frac{\mathrm{nint}(1p^*)}{1}, \frac{\mathrm{nint}(2p^*)}{2}, \ldots, \frac{\mathrm{nint}(np^*)}{n} \right), \tag{7.10}$$

where nint is the nearest integer (or round) function such that

$$\mathrm{nint}(x) = \begin{cases} \mathrm{floor}(x) & \text{if } x - \mathrm{floor}(x) < \mathrm{ceil}(x) - x, \\ \mathrm{ceil}(x) & \text{if } x - \mathrm{floor}(x) > \mathrm{ceil}(x) - x, \\ \mathrm{tbf}(x) & \text{if } x - \mathrm{floor}(x) = \mathrm{ceil}(x) - x. \end{cases} \tag{7.11}$$

Here tbf is a "tie-breaking function" and handles the case where $x - \mathrm{floor}(x) = \mathrm{ceil}(x) - x$. To ensure no bias, tbf is defined to use stochastic rounding

$$\mathrm{tbf}(x) = \begin{cases} \mathrm{floor}(x) & \text{with probability } \frac{1}{2}, \\ \mathrm{ceil}(x) & \text{with probability } \frac{1}{2}. \end{cases} \tag{7.12}$$

The difference between each term in S_2 and p^* is

$$\begin{aligned} S_3 &= \left(\frac{\mathrm{nint}(1p^*)}{1} - p^*, \frac{\mathrm{nint}(2p^*)}{2} - p^*, \ldots, \frac{\mathrm{nint}(np^*)}{n} - p^* \right) && (7.13) \\ &= \left(\frac{\mathrm{nint}(1p^*) - 1p^*}{1}, \frac{\mathrm{nint}(2p^*) - 2p^*}{2}, \ldots, \frac{\mathrm{nint}(np^*) - np^*}{n} \right) \text{ and,} && (7.14) \end{aligned}$$

the absolute value of each term in S_3 gives

$$S_4 = \left(\frac{|\mathrm{nint}(1p^*) - 1p^*|}{1}, \frac{|\mathrm{nint}(2p^*) - 2p^*|}{2}, \ldots, \frac{|\mathrm{nint}(np^*) - np^*|}{n} \right). \tag{7.15}$$

If S_4 converges to 0, then S_3 must converge to 0, and S_2 must converge to p^*. Additionally, the convergence rate of S_4 to 0 will equal both the convergence rate of S_3 to 0, and the convergence rate of S_2 to p^*.

If $p^* = 0$, then

$$\begin{aligned} S_2 &= (0, 0, \ldots, 0), \\ S_3 &= (0, 0, \ldots, 0), \\ S_4 &= (0, 0, \ldots, 0), \end{aligned}$$

where S_2, S_3, and S_4 converge immediately (or their convergence rates are infinity).

If $p^* = 1$, then

$$\begin{aligned} S_2 &= (1, 1, \ldots, 1), \\ S_3 &= (0, 0, \ldots, 0), \\ S_4 &= (0, 0, \ldots, 0), \end{aligned}$$

where S_2, S_3, and S_4 converge immediately (or their convergence rates are infinity).

If $0 < p^* < 1$, $i \in \mathbb{N}$, and $i \geq 1$, then for S_3

$$\min_{i,p^*}(\text{nint}(ip^*) - ip^*) = -0.5, \tag{7.16}$$

$$\max_{i,p^*}(\text{nint}(ip^*) - ip^*) = 0.5, \tag{7.17}$$

$$\text{nint}(ip^*) - ip^* \in [-0.5, 0.5]. \tag{7.18}$$

It follows that

$$\min_{i,p^*}\left(\frac{\text{nint}(ip^*) - ip^*}{i}\right) = \frac{-0.5}{i}, \tag{7.19}$$

$$\max_{i,p^*}\left(\frac{\text{nint}(ip^*) - ip^*}{i}\right) = \frac{0.5}{i}, \tag{7.20}$$

$$\frac{\text{nint}(ip^*) - ip^*}{i} \in \left[\frac{-0.5}{i}, \frac{0.5}{i}\right]. \tag{7.21}$$

For S_4,

$$\min_{i,p^*}(|\text{nint}(ip^*) - ip^*|) = 0.0, \tag{7.22}$$

$$\max_{i,p^*}(|\text{nint}(ip^*) - ip^*|) = 0.5, \tag{7.23}$$

$$|\text{nint}(ip^*) - ip^*| \in [0.0, 0.5]. \tag{7.24}$$

It follows that

$$\min_{i,p^*}\left(\frac{|\text{nint}(ip^*) - ip^*|}{i}\right) = \frac{0.0}{i}, \tag{7.25}$$

$$\max_{i,p^*}\left(\frac{|\text{nint}(ip^*) - ip^*|}{i}\right) = \frac{0.5}{i}, \tag{7.26}$$

$$\frac{|\text{nint}(ip^*) - ip^*|}{i} \in \left[\frac{0.0}{i}, \frac{0.5}{i}\right]. \tag{7.27}$$

It is important to recall that the lower and upper bounds of the terms in S_3 and S_4 are for any value $0 < p^* < 1$, and for any value $i \in \mathbb{N}$, $i \geq 1$. If p^* is set to a particular value, then these bounds may be tighter. For example, if $p^* = 1/3$, then $\text{nint}(ip^*) - ip^* \in [-1/3, 1/3]$ and $|\text{nint}(ip^*) - ip^*| \in [0, 1/3]$.

Consider the following facts about S_4:

1. The numerators of the terms in S_4 never converge, but are bounded.

2. The numerators of the terms in S_4 oscillate between their lower and upper bounds as the number of iterations increases. This means that given any iteration i_1, there exist iterations $i_2, i_3 > i_1$, where at i_2 the numerator of the term is approximately equal to its lower bound, and at i_3 the numerator of the term is approximately equal to its upper bound.

3. The sequence of upper bounds for S_4,

$$\left(\frac{0.5}{1}, \frac{0.5}{2}, \ldots, \frac{0.5}{\infty}\right), \tag{7.28}$$

is a positive sequence that converges to zero. Furthermore, this sequence converges sublinearly i.e.

$$\lim_{i\to\infty} \frac{|0.5/(i+1) - 0|}{|0.5/i - 0|} = \lim_{i\to\infty} \frac{i}{i+1} = 1, \tag{7.29}$$

and, more specifically, it converges logarithmically i.e.

$$\lim_{i\to\infty}\frac{|0.5/(i+2)-0.5/(i+1)|}{|0.5/(i+1)-0.5/i|}=\lim_{i\to\infty}\frac{i}{i+2}=1, \tag{7.30}$$

.

Given points 1 and 2, then the convergence rate of the terms in S_4 to zero must be equal to the convergence rate of its upper bound to zero (its lower bound is already zero). Considering this with point 3, it then follows that S_4 converges to zero, S_3 converges to zero, and S_2 converges to p^*, each with a convergence rate of at least c/i i.e.

$$\frac{|\mathrm{nint}(ip^*)-ip^*|}{i}\le\frac{c}{i},\forall i, \tag{7.31}$$

where c is a positive constant with a maximum value of 0.5. Thus we can say that the maximum convergence rate of an empirical probability is like $O(1/i)$.

In short, the claim is that S_1 cannot converge faster than $1/i$.

Proof 1. For any $\alpha\in\mathcal{A}$, its empirical probability in S_1, $\sum_{j=1}^{i}[\![\alpha_j=\alpha]\!]/i$, cannot converge to any probability, $0\le p\le 1$, faster than in S_2, $\mathrm{nint}(ip)/i$ where nint is the nearest integer (or round) function.

2. If $f(i)=D(S_2(i)||p)$ where D is the divergence, then

   (a) $f(i)\le 0.5/i$, and

   (b) $f(i)>c/i$ infinitely often where $0<c<0.5$.

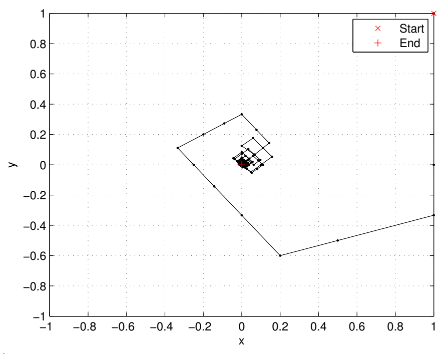
From point 2a it follows that $f(i)=O(1/i)$, and from point 2b it follows that $\nexists g(i):g(i)=o(1/i),f(i)=O(g(i))$.

Average Jensen-Shannon Divergence Metric

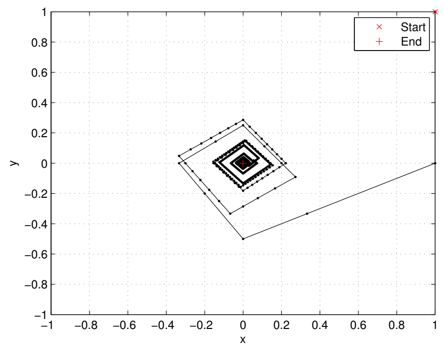
The maximum convergence rate of an empirical probability in an empirical distribution must also be the maximum convergence rate of the average Jensen-Shannon divergence metric between the empirical distribution and the distribution it is converging to (i.e. the distance cannot converge faster than any individual empirical probability can converge). Assuming that the maximum convergence rate for any empirical probability in an empirical distribution is like $O(1/i)$, where i is the iteration, then this must also be the maximum convergence rate for the average Jensen-Shannon divergence metric.

7.4.3 Matching Pennies

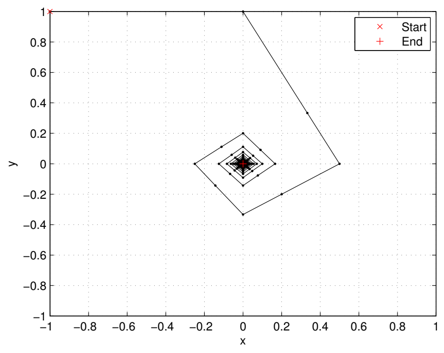
The results for matching pennies are shown in figures 7.1 and 7.3. They show that, in comparison to fictitious play, the empirical Nash convergence of each other method is faster. Specifically, comparing their average Jensen-Shannon divergence metrics, each method is converging optimally like $1/t$, whereas fictitious play is converging like $1/\sqrt{t}$. Similarly to fictitious play, their empirical distributions of plays cycle around the Nash equilibrium to some degree, with successive cycles getting smaller. However, the cycles of these methods get smaller more quickly. Change detection with or without fictitious play has cycles that get closer to the Nash equilibrium at an almost consistent rate, which is similar to fictitious play on its own. Sequence prediction with or without fictitious play has more irregular cycles than change detection or fictitious play on its own. For sequence prediction with fictitious play, there are jumps in how close its cycles are to the origin, which correspond to the sequence prediction method temporarily being used to make predictions.



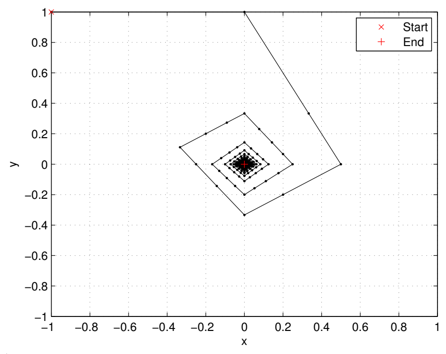
(a) Sequence prediction. At the start, $x = 1$, $y = 1$, and at the end, $x = 0$, $y = 0$. They are converging towards the Nash equilibrium at the centre.



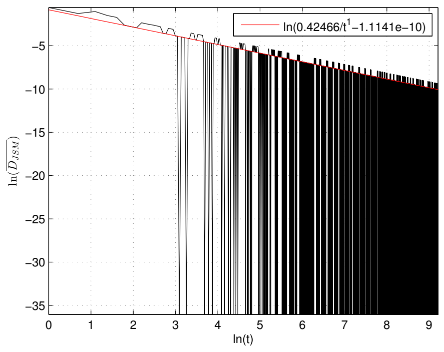
(b) Sequence prediction and fictitious play. At the start, $x = 1$, $y = 1$, and at the end, $x = -4 \times 10^{-4}$, $y = 1 \times 10^{-3}$. They are converging towards the Nash equilibrium at the centre.



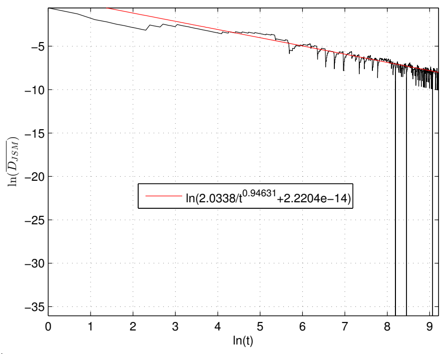
(c) Change detection. At the start, $x = -1$, $y = 1$, and at the end, $x = 2\times10^{-4}$, $y = 0$. They are converging towards the Nash equilibrium at the centre.



(d) Change detection and fictitious play. At the start, $x = -1$, $y = 1$, and at the end, $x = 4\times10^{-4}$, $y = -4\times10^{-4}$. They are converging towards the Nash equilibrium at the centre.

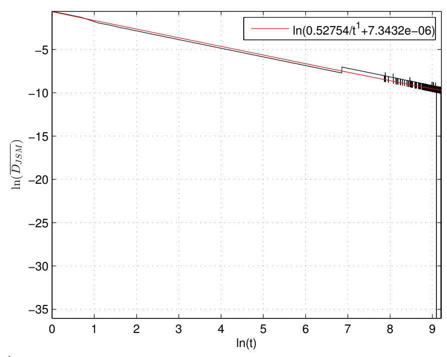


(e) Sequence prediction. The average Jensen-Shannon divergence metric is converging towards 0.

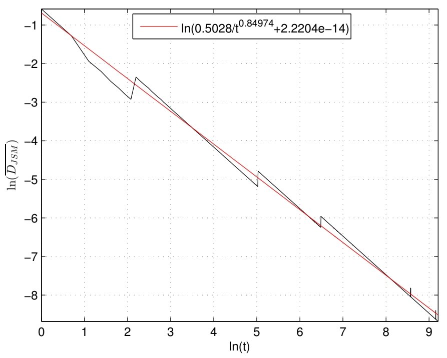


(f) Sequence prediction and fictitious play. The average Jensen-Shannon divergence metric is converging towards 0.

Figure 7.3



(g) Change detection. The average Jensen-Shannon divergence metric is converging towards 0.



(h) Change detection and fictitious play. The average Jensen-Shannon divergence metric is converging towards 0.

Figure 7.3: Sequence prediction with or without fictitious play and change detection with or without fictitious play each in self-play in matching pennies over 1×10^4 iterations. The parameters of the best-fit lines were calculated using MATLAB's Trust-Region-Reflective Least Squares algorithm with Bisquare weights *Least-Squares Algorithms*. Note that these graphs are not fully representative of the overall convergence speeds. For example, in generalised matching pennies, combining sequence prediction with fictitious play generally slightly speeds up its convergence rather than slowing it down as shown here.

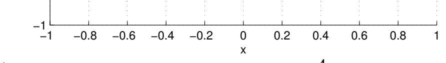
7.4.4 Generalised Matching Pennies

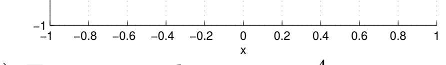
The results for a variety of two-player, two-action, normal-form games derived from generalised matching pennies are shown in Figure 7.4. The results are given for after 1×10^4 iterations, and after 1×10^5 iterations. They show that fictitious play has empirical Nash convergence in all of the games, which is expected theoretically. This is not the case for sequence prediction, which does not have empirical Nash convergence in most cases. In fact, the results for sequence prediction in matching pennies seem to be more of an exception rather than the rule. It seems to converge further away from a Nash equilibrium when at that Nash equilibrium at least one player has a strategy with a large magnitude. Sequence prediction and fictitious play improves on sequence prediction by having empirical Nash convergence in more cases. The cases where it does not are where at the Nash equilibrium at least one player has a strategy with a large magnitude. Change detection, like fictitious play, converges in all cases, but there are a few cases where it is still converging after 1×10^4 iterations. These cases are where one player's Nash equilibrium strategy is to be almost indifferent between its actions, and the other player's Nash equilibrium strategy is to be almost certain of its actions. Combining change detection with fictitious play causes it to converge in all cases after 1×10^4 iterations.

The results also show estimates for the mean empirical convergence rate, $\overline{b}$, and the mean asymptotic convergence distance from the Nash equilibria, $\overline{c}$, for each method. These estimates are calculated by fitting the equation

$$\overline{D_{JSM}} = \frac{a}{t^b} + c \tag{7.32}$$

to the results of each game after 1×10^4 iterations and after 1×10^5 iterations and finding the mean b and c parameters. Fictitious play has empirical convergence rates like $1/\sqrt{t}$, whereas the other methods have nearly optimal empirical convergence rates like $1/t$. Also for fictitious play and change detection with or without fictitious play, $\overline{c} = 0.00$, so they empirically converge to the Nash equilibria on average. Whereas for sequence prediction with or without fictitious play, $\overline{c} > 0$, so they sometimes empirically converge away from the Nash equilibria.

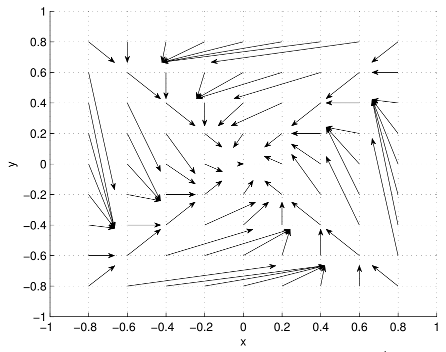




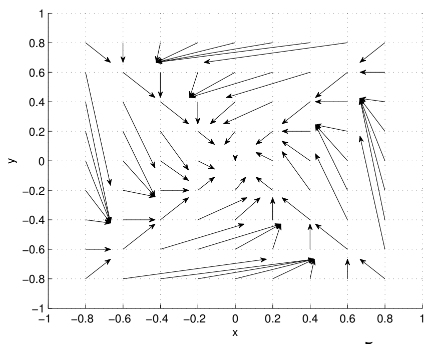
(a) Fictitious play, 1×10^4 iterations, $\overline{b} = 0.55$, $\overline{c} = 0.00$. Each average Jensen-Shannon divergence metric is converging towards 0.

(b) Fictitious play, 1×10^5 iterations, $\overline{b} = 0.54$, $\overline{c} = 0.00$. Each average Jensen-Shannon divergence metric is converging towards 0.

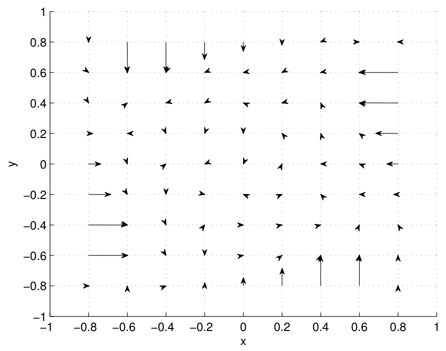
Figure 7.4



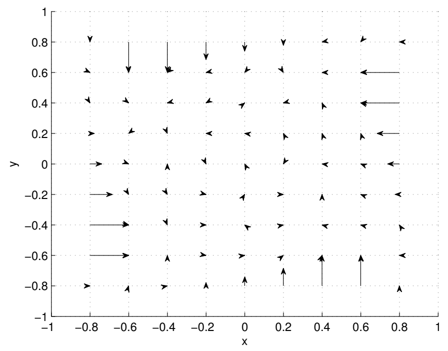
(c) Sequence prediction, 1×10^4 iterations, $\bar{b} = 0.93$, $\bar{c} = 0.09$. Each average Jensen-Shannon divergence metric is converging towards $c \geq 0$, where it tends to be larger if at least one player has a Nash equilibrium strategy with a large magnitude.



(d) Sequence prediction, 1×10^5 iterations, $\bar{b} = 0.98$, $\bar{c} = 0.09$. Each average Jensen-Shannon divergence metric is converging towards $c \geq 0$, where it tends to be larger if at least one player has a Nash equilibrium strategy with a large magnitude.

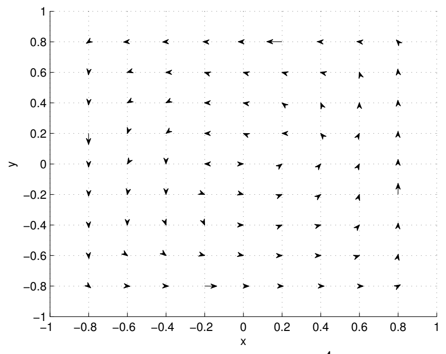


(e) Sequence prediction and fictitious play, 1×10^4 iterations, $\bar{b} = 0.95$, $\bar{c} = 0.01$. Most average Jensen-Shannon divergence metrics are converging towards 0. Some are converging towards $c > 0$ where at least one player has a Nash equilibrium strategy with a large magnitude.

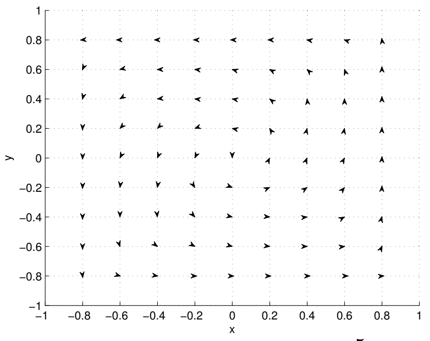


(f) Sequence prediction and fictitious play, 1×10^5 iterations, $\bar{b} = 0.98$, $\bar{c} = 0.01$. Most average Jensen-Shannon divergence metrics are converging towards 0. Some are converging towards $c > 0$ where at least one player has a Nash equilibrium strategy with a large magnitude.

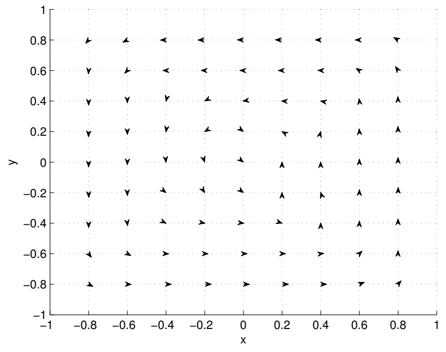
Figure 7.4



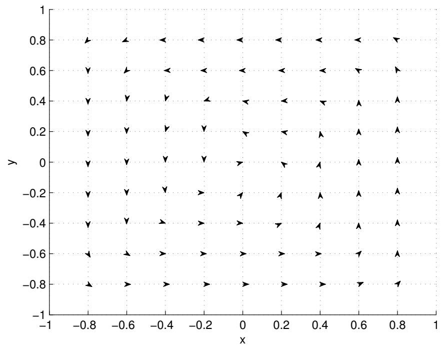
(g) Change detection, 1×10^4 iterations, $\overline{b} = 0.98$, $\overline{c} = 0.00$. Almost all average Jensen-Shannon divergence metrics are converging towards 0. Only a few are converging towards $c > 0$ where one player has a Nash equilibrium strategy near 0 and the other player does not.



(h) Change detection, 1×10^5 iterations, $\overline{b} = 0.99$, $\overline{c} = 0.00$. Each average Jensen-Shannon divergence metric is converging towards 0.



(i) Change detection and fictitious play, 1×10^4 iterations, $\overline{b} = 0.89$, $\overline{c} = 0.00$. Each average Jensen-Shannon divergence metric is converging towards 0.



(j) Change detection and fictitious play, 1×10^5 iterations, $\overline{b} = 0.80$, $\overline{c} = 0.00$. Each average Jensen-Shannon divergence metric is converging towards 0.

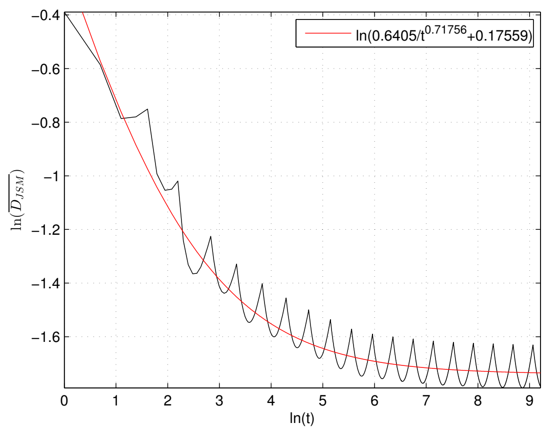
Figure 7.4: Convergence of the empirical distributions of plays of various methods in self-play in two-player, two-action, normal-form games derived from generalised matching pennies with Nash equilibria at positions $\{(x^*, y^*)|x^* \in \{-0.8, -0.7, \ldots, 0.8\}\}, y^* \in \{-0.8, -0.7, \ldots, 0.8\}\}$. Each arrow points from a Nash equilibrium position to the position of the method's empirical distribution of plays. Estimates for the mean convergence rate, $\overline{b}$, and the mean asymptotic convergence distance from the Nash equilibria, $\overline{c}$, are shown for each method after 1×10^4 iterations and after 1×10^5 iterations. This is calculated by fitting the equation $\overline{D_{JSM}} = a/t^b + c$ to the results of each game and taking the average of the b values as well as the average of the c values. The parameters of the best-fit lines were calculated using MATLAB's Trust-Region-Reflective Least Squares algorithm with Bisquare weights *Least-Squares Algorithms*.

Table 7.1: Estimates for the mean convergence rate, $\overline{b}$, and the mean asymptotic convergence distance from the Nash equilibria, $\overline{c}$, for each method after 1×10^4 iterations and after 1×10^5 iterations in two-player, two-action, normal-form games derived from matching pennies. This is calculated by fitting the equation $\overline{D_{JSM}} = a/t^b + c$ to the results of each game and taking the average of the b values as well as the average of the c values. The parameters of the best-fit lines were calculated using MATLAB's Trust-Region-Reflective Least Squares algorithm with Bisquare weights *Least-Squares Algorithms*.

Method	1×10^4 iterations		1×10^5 iterations	
	$\overline{b}$	$\overline{c}$	$\overline{b}$	$\overline{c}$
Fictitious play	0.55	0.00	0.54	0.00
Sequence prediction	0.93	0.09	0.98	0.09
Sequence prediction and fictitious play	0.95	0.01	0.98	0.01
Change detection	0.98	0.00	0.99	0.00
Change detection and fictitious play	0.89	0.00	0.80	0.00

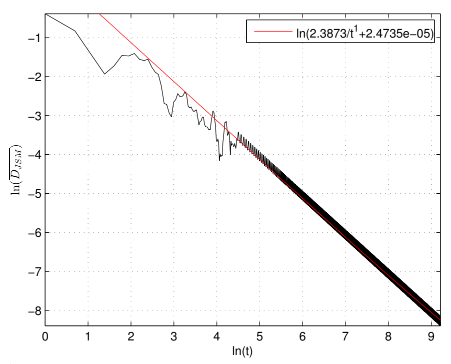
7.4.5 Shapley's Game

The results for Shapley's game are shown in Figure 7.5. They show that fictitious play does not have empirical Nash convergence. Its average Jensen-Shannon divergence metric decreases slightly but eventually oscillates around a value away from zero with constant amplitude and an ever increasing period. Change detection follows a similar pattern, except its oscillations decrease in amplitude until they eventually fade out, and its value is much closer to zero such that it essentially has empirical Nash convergence. Sequence prediction with or without fictitious play as well as change detection with fictitious play have empirical Nash convergence at a nearly optimal rate like $1/t$.

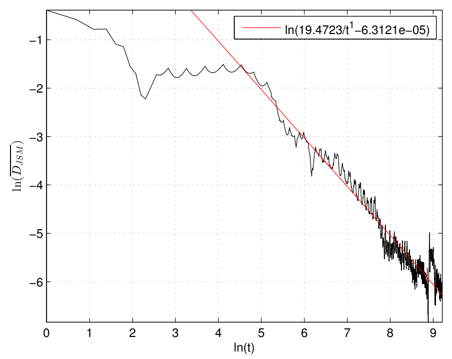


(a) Fictitious play. The average Jensen-Shannon divergence metric is oscillating around a value away from 0.

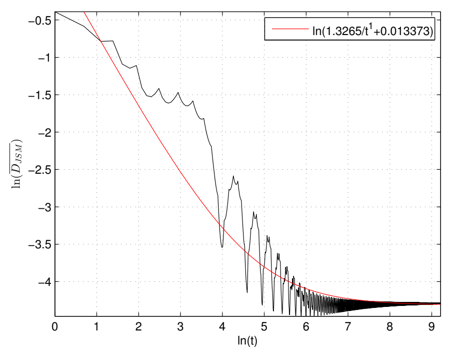
Figure 7.5



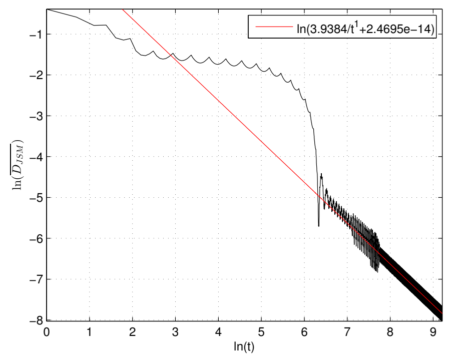
(b) Sequence prediction. The average Jensen-Shannon divergence metric is converging towards 0.



(c) Sequence prediction and fictitious play. The average Jensen-Shannon divergence metric appears to be converging towards 0.



(d) Change detection. The average Jensen-Shannon divergence metric is converging towards a value near 0.

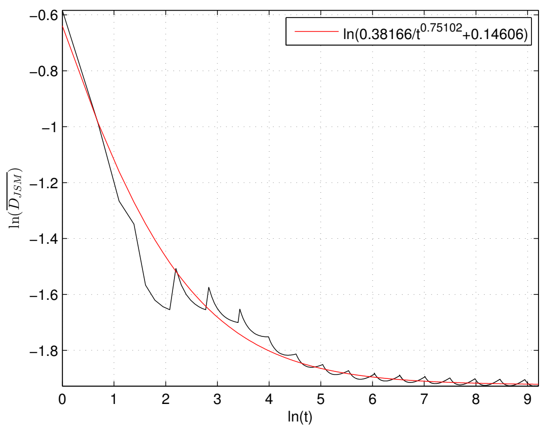


(e) Change detection and fictitious play. The average Jensen-Shannon divergence metric is converging towards 0.

Figure 7.5: Convergence of empirical distributions of plays of various methods in self-play in Shapley's game over 1×10^4 iterations. The parameters of the best-fit lines were calculated using MATLAB's Trust-Region-Reflective Least Squares algorithm with Bisquare weights *Least-Squares Algorithms*.

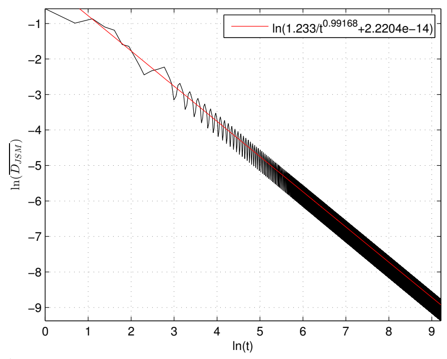
7.4.6 Jordan's Game

The results for Jordan's game are shown in Figure 7.6. They show that fictitious play does not have empirical Nash convergence. Its average Jensen-Shannon divergence metric oscillates around a value away from zero with constant amplitude and an ever increasing period. In contrast, each of the other methods converge at a rate like $1/t$.

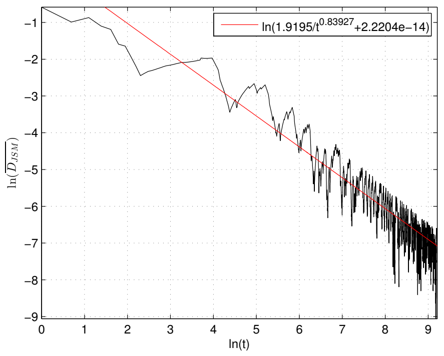


(a) Fictitious play. The average Jensen-Shannon divergence metric is oscillating around a value away from 0.

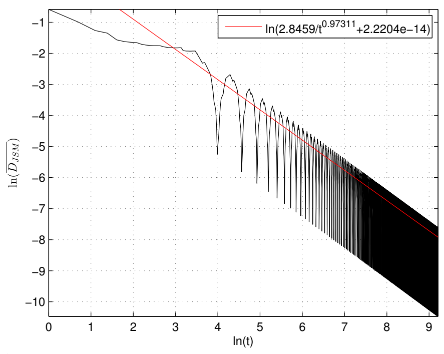
Figure 7.6



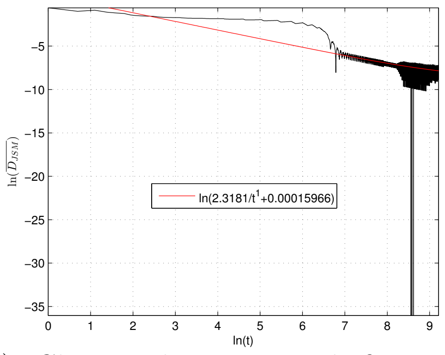
(b) Sequence prediction. The average Jensen-Shannon divergence metric is converging towards 0.



(c) Sequence prediction and fictitious play. The average Jensen-Shannon divergence metric is converging towards 0.



(d) Change detection. The average Jensen-Shannon divergence metric is converging towards 0.



(e) Change detection and fictitious play. The average Jensen-Shannon divergence metric is converging towards 0.

Figure 7.6: Convergence of the empirical distributions of plays of various methods in self-play in Jordan's game over 1×10^4 iterations. The parameters of the best-fit lines were calculated using MATLAB's Trust-Region-Reflective Least Squares algorithm with Bisquare weights *Least-Squares Algorithms*.

7.5 Chapter Summary

In this chapter, two new variants of fictitious play have been proposed, each of which assume that the opponents have dynamic strategies. The first variant uses sequence prediction to predict an opponent's strategy based on different contexts of its most recent actions and its empirical distributions of plays that have occurred after these contexts. The second variant uses change detection to infer a distribution over possible changepoints in an opponent's strategy and uses this distribution to predict its strategy. Each variant, like fictitious play, plays a pure best-response strategy to its predicted opponent strategies. The main advantage of sequence prediction and change detection methods in this context, is that by assuming that the opponent uses a dynamic strategy, they can potentially convergence more quickly to solution concepts. One way they can do this is by helping to avoid correlated play, which otherwise could make empirical distributions of plays cycle around Nash equilibria, as well as the expected rewards of strategies equal to empirical distributions of plays not be obtained.

Experiments compared the convergence in self-play of the empirical distributions of plays of fictitious play and the proposed variants to mixed strategy Nash equilibria. The results show that the proposed variants converge faster than fictitious play in all of the tested games. However, sequence prediction does not always converge exactly to the Nash equilibria in games derived from generalised matching pennies. Specifically, whilst fictitious play and change detection always converge in the tested generalised matching pennies games, sequence prediction does not converge in most cases. Combining sequence prediction with fictitious play in these generalised matching pennies games decreases its convergence distance from Nash equilibria and increases the estimate of its mean convergence speed. Combining change detection with fictitious play in these generalised matching pennies games decreases its convergence distance from Nash equilibria in a handful of cases given 1×10^4 iterations, but decreases the estimate of its mean convergence speed. The results also show that, unlike fictitious play, the proposed variants and the hybrid algorithms converge to the Nash equilibria in Shapley's and Jordan's games, which are known to be difficult. Overall, this shows that whilst sequence prediction is somewhat unstable, change detection has a better self-play performance than fictitious play within these games.

Future work could investigate the range of circumstances under which the proposed variants converge. It could also investigate why the proposed variants do or do not converge. The answer may be based on the amount of history that the agents are using, if this is too low, then the agents may have insufficient resolution to accurately predict an opponent's strategy. For example, using a sliding window of size one, it would always appear as if the opponent will repeat their last action. The ideas and results in this chapter should generalise to similar normal-form games and situations where fictitious play has been successful like in limit Texas hold'em [Dudziak, 2006; Fellows, 2010].

Chapter 8

Conclusions and Future Work

This thesis has investigated decision-making with imperfect information when facing opponents whose actions can affect our rewards, and whose strategies may be based on a memory of interaction, or may be changing, or both. Ideally, we want to maximise an agent's rewards in any situation. This thesis has looked at situations with hidden information represented through games. Almost all of these games have just two players. There are numerous advantages to using games, the main advantage is that they allow you to focus on developing the players rather than worrying about how the games work. Additionally, they can model many real-world problems. The focus has been on modelling dynamic opponents in hidden information games and learning high-reward strategies based on these models.

This chapter summarises what has been learnt from the research presented in this thesis. It also discusses the contributions of this research to the field of machine learning, as well as possible directions for future research.

8.1 Conclusions

This section summarises the contributions of this research and presents what has been learnt about modelling opponents with dynamic strategies and taking advantage of these opponent models.

8.1.1 First Approach: Sequence Prediction, Reinforcement Learning, and Lookahead

The first approach in Chapter 4 looked at small simultaneous-move games, where the hidden information is the opponent's action, which is always revealed after the player and the opponent act. The player's, possibly delayed, rewards for its actions were learnt with a popular reinforcement learning algorithm, namely Q-Learning. By predicting the player's rewards for its future actions using Q-Learning, and the probabilities that the opponent would allow the player to reach those actions using sequence prediction methods, this allowed explicit lookahead (tree search) to be used to predict the sequence of actions that would maximise the player's expected cumulative rewards. This approach empirically maximised the player's rewards against several opponents with memory-based strategies, and gained higher rewards than opponents using popular and state-of-the-art reinforcement learning algorithms.

8.1.2 Second Approach: Expectation-Maximisation, Sequence Prediction, and No-Regret Learning

The second approach in Chapter 5 looked at medium-size sequential-move games where the opponent's strategy depends on its own hidden information that is part of the game, which is not always revealed at the end of a game. Knowing the opponent's hidden information was necessary to model its strategy and so, if necessary, it was predicted using an online expectation-maximisation algorithm. By being able to predict the opponent's actions, games could be simulated games against the opponent model. The player's strategy was updated using only its rewards in games against the opponent and simulated games against the opponent model with a state-of-the-art no-regret learning algorithm, namely OS-MCCFR. Experiments in simplified poker games showed that this approach significantly improved the average payoff per game of OS-MCCFR against popular and state-of-the-art reinforcement learning methods.

The reason that the opponent's hidden information is inferred to model its strategy is because it may base its actions on its hidden information. For example, in poker the opponent's cards, which are not revealed unless a showdown occurs, are what it would base its betting decisions on if it was a good player. Ignoring its hidden information would effectively assume that its actions do not depend on it. This can be a very bad assumption. For example, a good opponent would act differently with a royal flush than with a high card. Therefore, it is important to try to predict the opponent's hidden information to better model its strategy.

The overall approach works as follows. At the end of a game, the online expectation-maximisation algorithm predicts the probability of each possible instance of the opponent's hidden information. This allows an instance of the opponent's hidden information to be sampled from this distribution, which is then assumed to represent the opponent's true hidden information. The opponent model is then updated using the sequence prediction method with the prediction of the opponent's hidden information and observations of its actions.

The empirical results found that this approach improved the average payoff per game of OS-MCCFR in die-roll poker and Rhode Island hold'em. However, it was also found that the online expectation-maximisation algorithm gave less of an improvement in the latter, which is probably due to it having more hidden information. More hidden information means that there is an increased chance that the opponent will behave similarly for different instances of hidden information, making it more difficult for expectation-maximisation to distinguish the opponent's hidden information based on the opponent's behaviour.

8.1.3 Sequence Prediction Methods

In chapters 4, 5, and 7 sequence prediction methods have been adapted and applied to build explicit opponent models to be used to increase agents' rewards. Empirical results in various games have shown that using opponent models built with sequence prediction methods lead to higher rewards than using opponent models built using empirical probabilities against opponents with memory-based or changing strategies. For example, in the iterated prisoner's dilemma, the tit-for-tat strategy can only be modelled by learning that it copies its opponent's most recent action. In other words, tit-for-tat's strategy is conditioned on a memory of its opponent's most recent action, which sequence prediction methods are well suited to learn.

Many sequence prediction methods have been looked at, each with its own set of features. Although any single sequence prediction method has not been found that is objectively better than the others, there are some features that are arguably more important than others for the purpose of building explicit opponent models. One feature that all sequence prediction methods share is the ability to model conditional strategies. Other than this feature, the most important features are arguably as follows. Firstly, context pruning, which is the ability to discard contextual information that is believed to be obsolete as this can forget data that becomes irrelevant if an opponent's strategy changes. Secondly, context blending, which is the ability to blend predictions from contexts with different lengths as this gives the possibility for all possible contexts to make a contribution. Finally, context length, as a longer context lengths allows for patterns to be recognised for longer sequences.

8.1.4 Change Detection Methods

Change detection methods have also been examined in chapters 6 and 7, which like sequence prediction methods, can model changing opponent strategies. Variations of three state-of-the-art change detection methods were examined in chapter 6, which included some modifications proposed in this thesis to allow them to learn an opponent's strategy. These change detection methods were empirically compared in their abilities to model changing opponent strategies represented as discrete probability distributions. It was found that the Bayesian Change Point Detection (BayesCPD) methods generally maintain the most accurate distributions, which can be used to produce the most rewarding counter strategies. This method was later used in chapter 7 to try to improve on the convergence in self-play of fictitious play.

8.1.5 Self-Play Convergence of Empirical Distributions of Plays

Finally in chapter 7 this thesis looked at the self-play convergence of the empirical distributions of plays of candidate sequence prediction and change detection methods as well as two hybrid methods, which combined fictitious play with these methods, to mixed Nash equilibrium strategies. Unlike fictitious play, sequence prediction and change detection methods can recognise correlations between actions, which allows their empirical distributions of plays to converge to Nash equilibrium strategies in self-play in situations where fictitious play does not converge. It was shown that, for this type of convergence, all of these methods converge faster than fictitious play in particular games. However, although the change detection method always converged to the Nash equilibria, in many cases, the sequence prediction method did not. Combining the sequence prediction method with fictitious play reduced this number of cases but did not get rid of all of them. Combining the change detection method with fictitious play caused it to converge in fewer iterations in some cases. Additionally, unlike fictitious play, both the sequence prediction and change detection methods converged to the Nash equilibria in Shapley's and Jordan's games, which are considered difficult.

8.2 Future Work

This section outlines some possible directions for future research based on what has been investigated.

8.2.1 Safe Strategy

Building an explicit opponent model without any prior information typically takes a lot of time as the opponent's strategy scales with their number of information sets (decision points). Even if the opponent modelling algorithm can quickly learn the opponent's strategy for particular decision points, there may be many decision points. Unless it is lucky, the opponent model will probably initially be inaccurate due to very little data being observed. Therefore, the agent's best option may be to play a safe strategy until enough data has been gathered to be confident that the opponent model is reasonably accurate. This is not a new concept, previous work has used approximate Nash equilibrium strategies whilst building their opponent models[1]. However, a Nash equilibrium strategy is, in general, difficult to compute. Instead, it may be better to use alternative safe strategies that are easier to compute such as ϵ-Nash equilibrium strategies or dominant strategies. Another aspect to consider is how to shift from the safe strategy to a more exploitative strategy that takes advantage of the opponent model. The shift should probably be proportional to the accuracy of the opponent model but this can be difficult to measure. Additionally, the shift should probably be reversible in case the opponent model becomes inaccurate, perhaps due to the opponent changing its strategy. One potential problem with using safe strategies is that if the opponent is adaptive, then by playing a safe strategy with little exploitability you may end up training the opponent to also play a safe strategy

[1]Bard et al., 2013; Ganzfried and Sandholm, 2011; Johanson and Bowling, 2009; Johanson, Zinkevich, and Bowling, 2008; Ponsen, Lanctot, and Jong, 2010.

with little exploitability. For example, if you play a Nash equilibrium strategy, then the opponent may learn to play its Nash equilibrium strategy from that Nash equilibrium. The opponent model would also become less useful as the opponent adapts to become less exploitable.

8.2.2 Opponent Strategy Space

Many situations offer an agent a huge strategy space, which is too large to explore within a reasonable amount of time. To deal with this, an agent would often use an abstraction. The abstraction serves to reduce its strategy space to a more manageable size. In the experiments in this thesis, it was assumed that the agents use the same levels of abstraction. This raises the question of what happens if they use different levels of abstraction and specifically what effects would this have on an opponent model. Ideally, the opponent model should have a strategy space that is equal to the opponent's strategy space. If the strategy space is larger, then time will be wasted learning different parts of the opponent's strategy that are actually the same. If the strategy space is smaller, then the opponent model may be more inaccurate as it is unable to distinguish between strategically different situations. Moreover, it may not just be about the size of the strategy space but also the interpretation. If the opponent model sees a different strategy space than the opponent, then this could also cause the opponent model to be inaccurate.

8.2.3 Sequence Prediction Methods

Out of all the sequence prediction methods used, one has not been found that is objectively better than all of the others at predicting opponent actions. However, there are sequence prediction methods that are arguably better than the sequence prediction methods used in this thesis when being used for compression. In particular, the PAQ series of data compression algorithms currently have the best compression ratios on a variety of benchmarks [*Data Compression Programs*]. This suggests that their sequence prediction methods may be superior. If their sequence prediction methods could be extracted and adapted efficiently, then they might be able to be used to model the opponent more effectively. Other sequence prediction methods that may improve the opponent model are a stochastic memoizer [Wood et al., 2009] and optimised versions of prediction by partial matching. Many machine learning tasks can be (re)stated as sequence prediction problems including pattern recognition, classification, and time-series forecasting. Therefore, it is possible that advances in these related fields could be adapted as, or used to improve, sequence prediction methods.

8.2.4 Continuous Values

This thesis has assumed that an opponent's strategy is a collection of discrete probability distributions, one for each of its decision points. This is because in the games this thesis has considered, each of the opponent's decision points have always involved a discrete set of choices. However, many real-world scenarios involve decisions with a large, or infinite, number of choices. For example, choosing a speed for movement could take any value real value. For a continuous number of choices a discrete sequence prediction method will generally be unable to accurately predict the exact choice because there are so many possibilities. One way to deal with this is to discretise the choices by grouping similar choices together like how an abstraction groups similar states together. Instead of choosing from all possible speeds for movement, a discrete number of choices could be given such as fast, regular, or slow. Future work could look at applying opponent modelling based on sequence prediction or change detection to games or situations with continuous values.

8.2.5 N-player games

The focus of this thesis has been on two-player games. However, real-world decision-making problems often involve many decision-makers. Therefore, one way to further this research would be to try to apply its ideas to n-player games. In this case, the main challenge of an n-player game is that it would require each of the agents to keep track of n opponent models. It would take a lot more space to store and a lot more time to process n opponent models as opposed to just one opponent model especially if the game is large. However, one advantage of the approaches in this thesis is that they do not need to be modified to be applied to n-player games.

An interesting direction would be to investigate ways to take advantage of multiple opponent models. Perhaps they could be used to manipulate the opponents to play against each other. This could weaken the opponents and make an agent relatively stronger. It is reasonable to assume that as the number of opponents increases, the value of modelling each individual opponent decreases because each opponent would have less influence on the game (if the game is fair). Also, as already mentioned, more opponent models would need more space to store and more time to process. Therefore, in cases with many opponents it may be more beneficial to model groups rather than individuals, or to ignore certain opponents completely, or both.

8.2.6 Learning Objectives

The empirical results in this thesis show that the proposed approaches achieve higher rewards than comparison agents. However, the properties of these approaches have not been fully analysed in terms of those discussed in Section 2.2. Most multi-agent learning algorithms try to satisfy the two properties outlined by Bowling and Veloso, 2002, which are rationality, converging to a best-response strategy against a stationary opponent, and convergence, converging to a stationary strategy against a set of rational players. The empirical results have shown that the first approach can converge to a best-response strategy against an opponent using a memory-based strategy. Also, it is known that with the second approach, assuming its opponent model is accurate, through using the OS-MCCFR algorithm it will, in expectation, converge to a best-response strategy against an opponent with a stationary strategy.

It is unknown if the approaches in this thesis will converge to stationary strategies against all rational opponents. Although convergence against any set of rational players cannot be proven, a usual minimum but not sufficient test is to check convergence in self-play. It would be interesting to investigate under what circumstances the approaches in this thesis, or variations of them, can converge in self-play. In particular, one could look at what happens when both agents are using sequence prediction methods, or change detection methods, or both, to model each other. For example, it is well known that the empirical distribution of plays of fictitious play will converge to a Nash equilibrium in self-play in various games. This promotes the question of if the same can be said for sequence prediction and change detection methods. The preliminary experiments in Chapter 7 show that sequence prediction and change detection can convergence in self-play in some games.

8.2.7 Real-world Problems

Games are very useful for developing decision-making agents, mainly because they allow you to focus on developing the learning agent rather than the game itself. However, it would be interesting to test the approaches in this thesis on real-world problems. One potential application could be peer-to-peer networking, where peer negotiation strategies for uploading/downloading are similar to those used in the prisoner's dilemma, specifically Cohen, 2003 states that the BitTorrent protocol uses tit-for-tat for Pareto efficiency. Thus, the approach in Chapter 4 could be adapted and applied in this setting to try to effectively respond in agent interactions in the BitTorrent environment.

8.2.8 Prior Knowledge

This thesis has considered agents that develop strategies based only on observations of their own rewards and on observations of other agents' actions. Whilst learning without any prior knowledge is an interesting problem, prior knowledge can significantly increase an agent's rewards. Therefore, if the goal is to maximise an agent's rewards by any means necessary, then the agent should be given as much prior knowledge as possible that may help it achieve this goal. For example, if an agent was given prior knowledge that its opponent in an iterated prisoner's dilemma game was tit-for-tat, then it would know that it should always cooperate to gain the highest reward. Another avenue for future work could then look at the value of different types of prior knowledge with regard to agent performance.

8.2.9 Implicit Modelling

A focus in this work has been on creating and using explicit opponent models that predict the actions of an opponent at its decision points. However, there are two major problems with explicit opponent modelling. Firstly, the number of observations required to learn the opponent's strategy depends on the degrees of freedom in the opponent model which, must grow with the size of the opponent's strategy space. As the number of opponent information sets increases, the number of observations required to learn the opponent's strategy will grow even faster. Secondly, even if the opponent's strategy is learnt, the agent still needs to compute a best-response strategy which, can be brittle if there is any modelling error [Johanson, Zinkevich, and Bowling, 2008].

In this thesis, the first problem was handled using abstraction, reducing the number of information sets until the opponent's strategy could be learnt in a reasonable time. For the second problem, the approaches in this thesis try to use "good" responses, learnt through lookahead or simulations, instead of best-responses. Despite this, time is still a factor in that hundreds, thousands, or even tens-of-thousands of games are still required to learn high-reward strategies. Implicit opponent modelling can avoid this by letting an agent quickly learn to exploit an opponent, perhaps whilst an explicit opponent model is being built. Implicit opponent modelling builds a portfolio of counter-strategies offline and estimates how well each strategy in the portfolio performs against the opponent online. Based on these performances, the agent can then choose the best strategy, or combination of strategies, from their portfolio. The advantage of this approach is that its number of parameters does not grow with the opponent's strategy space. Implicit modelling may give higher expected rewards than explicit modelling when there is little data, such as in the early games, or when the available data might be unreliable, like after the opponent has switched its strategy. Whereas explicit modelling may give higher expected rewards with lots of reliable data, such as in the late games when agent strategies are more likely to have converged. This is because with an explicit opponent model you are trying to replicate the opponent's strategy exactly, which could allow a perfect counter-strategy to be built. With implicit modelling you are forced to rely on your portfolio of counter-strategies which, may not contain a perfect counter-strategy.

8.2.10 Sequence Prediction and Change Detection

Chapter 6 empirically compared a variety of change detection methods and discussed how they could be used to improve sequence prediction methods. In particular, sequence prediction methods assume that their conditional probability distributions are stationary. Whilst some sequence prediction methods have ways to handle changing distributions by, for example, pruning distributions that become "unpredictive", change detection methods offer an alternative approach that is arguably more focused because they directly try to detect changes and allow them to be handled. Future work could look at using change detection methods to detect changes in and update the context-based distributions of sequence prediction methods.

Bibliography

2009 World RPS Championships — World RPS Society. `http://www.worldrps.com/world-championships`. Accessed: 12/03/2014.

Abdallah, Sherief and Victor R. Lesser (2008). “Non-Linear Dynamics in Multiagent Reinforcement Learning Algorithms”. In: *AAMAS (3)*, pp. 1321–1324.

Adams, Ryan Prescott and David J.C. MacKay (2007). *Bayesian Online Changepoint Detection.*

Allis, Victor L. (1994). “Searching for Solutions in Games and Artificial Intelligence”. PhD thesis. University of Limburg.

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer (2002). “Finite-time Analysis of the Multiarmed Bandit Problem”. In: *Machine Learning* 47, pp. 2–3.

Awheda, Mostafa D. and Howard M. Schwartz (2013). “Exponential Moving Average Q-Learning Algorithm”. In: *IEEE ADPRL.*

Backgammon Programming. `http://www.bkgm.com/rgb/rgb.cgi?view+782`. Accessed: 10/10/2013.

Baker, R.J.S. and P.I. Cowling (2007). “Bayesian Opponent Modeling in a Simple Poker Environment”. In: *IEEE Symposium on Computational Intelligence and Games.*

Banerjee, Bikramjit and Jing Peng (2003). “Adaptive Policy Gradient in Multiagent Learning”. In: *AAMAS.*

Bard, Nolan et al. (2013). “Online Implicit Agent Modelling”. In: *Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent Systems.*

Bifet, Albert and Ricard Gavaldà (2007). “Learning from Time-Changing Data with Adaptive Windowing”. In: *2007 SIAM International Conference on Data Mining (SDM07).*

Billings, Darse et al. (1998). “Opponent Modeling in Poker”. In: *Proceedings of the 15th National Conference on Artificial Intelligence.*

Bowling, Michael (2005). “Convergence and No-Regret in Multiagent Learning”. In: *NIPS 17*.

Bowling, Michael and Manuela Veloso (2001). “Convergence of Gradient Dynamics with a Variable Learning Rate”. In: *Proceedings of the 18th International Conference on Machine Learning*, pp. 27–34.

— (2002). “Multiagent Learning Using a Variable Learning Rate”. In: *Artificial Intelligence* 136, pp. 215–250.

Bowling, Michael et al. (2015). “Heads-up limit holdem poker is solved”. In: *Science* 347.6218, pp. 145–149.

Broeck, Guy, Kurt Driessens, and Jan Ramon (2009). “Monte-Carlo Tree Search in Poker Using Expected Reward Distributions”. In: *Proceedings of the 1st Asian Conference on Machine Learning: Advances in Machine Learning*.

Brown, G. W. (1951). “Activity Analysis of Production and Allocation”. In: ed. by T. J. Koopmans. New York: Wiley. Chap. Iterative Solutions of Games by Fictitious Play, pp. 374–376.

Butterworth, John M. (2010). “Stability of Gradient-Based Learning Dynamics in Two-Agent Imperfect-Information Games”. PhD thesis. University of Manchester.

Butterworth, John M. and Jonathan L. Shapiro (2009). “Stability of Learning Dynamics in Two-Agent, Imperfect Information Games”. In: *Proc. of the 10th ACM SIGEVO workshop on FOGA*. ACM.

Cappé, Olivier and Eric Moulines (2008). “Online EM Algorithm for Latent Data Models”. In: *Journal of the Royal Statistical Society* 71, pp. 593–613.

Cepheus Poker Project. `http://poker.srv.ualberta.ca/about`. Accessed: 20/03/2015.

Chen, Xi, Xiaotie Deng, and Shang-Hua Teng (2009). “Settling the Complexity of Computing Two-Player Nash Equilibria”. In: *J. ACM* 56, 14:1–14:57.

Chess Programming WIKI – Alan Turing (2014). `http://chessprogramming.wikispaces.com/Alan+Turing`. Accessed: 09/07/2014.

Chien, Steve et al. (2000). “A Comparison of Coordinated Planning Methods for Cooperating Rovers”. In: *Proceedings of the 4th International Conference on Autonomous Agents*, pp. 100–101.

Chinook vs. the Checkers Champ - Top 10 Man-vs.-Machine Moments - TIME. `http://content.time.com/time/specials/packages/article/0,28804,2049187_2049195_2049286,00.html`. Accessed: 10/10/2013.

Cohen, Bram (2003). *Incentives Build Robustness in BitTorrent.*

Conitzer, Vincent and Tuomas Sandholm (2008). "New complexity results about Nash equilibria". In: *Games and Economic Behavior* 63, pp. 621–641.

Crawford, Vincent P. (1974). "Learning the Optimal Strategy in a Zero-Sum Game". In: *Econometrica* 42, pp. 885–891.

— (1985). "Learning Behavior and Mixed-Strategy Nash equilibria". In: *Journal of Economic Behavior & Organization* 6, pp. 69–78.

— (1989). "Learning and Mixed-Strategy Equilibria in Evolutionary Games". In: *Journal of Theoretical Biology* 140, pp. 537–550.

CrazyStone at Sensei's Library. http://senseis.xmp.net/?CrazyStone. Accessed: 10/10/2013.

Daskalakis, Constantinos, Paul W. Goldberg, and Christos H. Papadimitriou (2006). "The Complexity of Computing a Nash Equilibrium". In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing.*

Dasu, Tamraparni et al. (2006). "An Information-Theoretic Approach to Detecting Changes in Multi-Dimensional Data Streams". In: *Proceedings of the Symposium on the Interface of Statistics, Computing Science, and Applications.*

Data Compression Programs. http://mattmahoney.net/dc/. Accessed: 25/04/2014.

Davidson, Aaron et al. (2000). "Improved Opponent Modeling in Poker". In: *International Conference on Artificial Intelligence.*

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society* 39, pp. 1–38.

Devroye, Luc (1986). *Non-Uniform Random Variate Generation.* Springer-Verlag New York Inc.

Dudziak, William (2006). "Using Fictitious Play to Find Pseudo-Optimal Solutions for Full-Scale Poker". In: *ICAI.*

Fearnhead, Paul and Zhen Liu (2007). "On-line Inference for Multiple Change Points Problems". In: *Journal of the Royal Statistical Society B* 69, pp. 589–605.

Fellows, Ian (2010). "Pseudo-Optimal Solutions to Texas Holdem Poker with Improved Chance Node Abstraction". http://www.deducer.org/pmwiki/uploads/Main/fell_omen.pdf.

Fernandes, Andrew D. (2008). *Site-Specific Relative Evolutionary Rates.* `http://www.fernandes.org/txp/?c=math`. Blog.

Fernandes, Andrew D. and William R. Atchley (2008). "Site-specific evolutionary rates in proteins are better modeled as non-independent and strictly relative". In: *Bioinformatics* 24, pp. 2177–2183.

Frigyik, Bela A., Amol Kapila, and Maya R. Gupta (2010). *Introduction to the Dirichlet Distribution and Related Processes.* Tech. rep. Department of Electrical Engineering University of Washington.

Fudenberg, Drew and David M. Kreps (1993). "Learning Mixed Equilibria". In: *Games and Economic Behaviour* 5, pp. 320–367.

Fudenberg, Drew and David K. Levine (1998). *The Theory of Learning in Games.* The MIT Press.

Galla, Tobias and J. Doyne Farmer (2013). "Complex Dynamics in Learning Complicated Games". In: *Proc. of the National Academy of Sciences* 110.4, pp. 1232–1236.

Gama, Joo et al. (2004). "Learning with Drift Detection". In: *SBIA Brazilian Symposium on Artificial Intelligence.*

Ganzfried, Sam and Tuomas Sandholm (2011). "Game Theory-Based Opponent Modeling in Large Imperfect-Information Games". In: *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems.*

Gers, Felix A., Nicol N. Schraudolph, and Jürgen Schmidhuber (2002). "Learning Precise Timing with LSTM Recurrent Networks". In: *JMLR* 3, pp. 115–143.

Gopalratnam, Karthik and Diane J. Cook (2003). "ActiveLezi: An Incremental Parsing Algorithm for Sequential Prediction". In: *Proceedings of the 16th International Florida AI Research Society Conference*, pp. 38–42.

Hahn, Sunku (1999). "The Convergence of Fictitious Play in 3 x 3 Games with Strategic Complementarities". In: *Economics Letters* 64.1, pp. 57–60.

Hofbauer, Josef (1995). "Stability for the Best Response Dynamics". Preprint Vienna 1994. Revised Version Budapest.

IBM computer Watson wins Jeopardy clash — Technology — theguardian.com. `http://www.theguardian.com/technology/2011/feb/17/ibm-computer-watson-wins-jeopardy`. Accessed: 10/10/2013.

IBM100 - Deep Blue. `http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/`. Accessed: 10/10/2013.

Jensen, Steven et al. (2005). “Non-stationary Policy Learning in 2-player Zero Sum Games”. In: *Proceedings of the 20th National Conference on Artificial Intelligence.*

Johanson, Michael and Michael Bowling (2009). “Data Biased Robust Counter Strategies”. In: *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics.*

Johanson, Michael, Martin Zinkevich, and Michael Bowling (2008). “Computing Robust Counter-Strategies”. In: *Advances in Neural Information Processing Systems 20.* MIT Press.

Johanson, Michael Bradley (2007). “Robust Strategies and Counter-Strategies: Building a Champion Level Computer Poker Player”. MA thesis. University of Alberta.

Jordan, J. S. (1993). “Three Problems in Learning Mixed-Strategy Nash Equilibria”. In: *Games and Economic Behaviour* 5, pp. 368–386.

Knoll, Byron (2009). *Text Prediction and Classification Using String Matching.*

Knuth, Donald E., James H. Morris, and Vaughan R. Pratt (1977). “Fast Pattern Matching in Strings”. In: *SIAM Journal on Computing* 6, pp. 323–350.

Koller, Daphne, Nimrod Megiddo, and Bernhard von Stengel (1994). “Fast Algorithms for Finding Randomized Strategies in Game Trees”. In: *Proceedings of the 26th ACM Symposium on the Theory of Computing*, pp. 750–759.

Korb, Kevin B., Ann E. Nicholson, and Nathalie Jitnah (1999). “Bayesian Poker”. In: *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence.*

Krishna, Vijay (1992). “Learning in Games with Strategic Complementarities”. Harvard Business School Working Paper 92-073.

Kuhn, Harold W. (1953). “Extensive games and the problem of information”. In: *Annals of Mathematics Studies* 28, pp. 193–216.

Laird, Philip and Ronald Saul (1994). “Discrete Sequence Prediction and Its Applications”. In: *Machine Learning* 15, pp. 43–68.

Lanctot, Marc et al. (2009). “Monte Carlo Sampling for Regret Minimization in Extensive Games”. In: *Advances in Neural Information Processing Systems 22.*

Lanctot, Marc et al. (2012). “No-Regret Learning in Extensive-Form Games with Imperfect Recall”. In: *Proceedings of the 29th International Conference on Machine Learning.*

Least-Squares Algorithms. `http://www.mathworks.com/help/optim/ug/least-squares-model-fitting-algorithms.html`. Accessed: 13/04/2014.

Lempel and Ziv (1978). *Compression of Individual Sequences via Variable-Rate Coding.*

Leslie, David S. (2003). "Convergent Multiple-Timescales Reinforcement Learning Algorithms in Normal Form Games". In: *AAP* 13, pp. 1231–1251.

Liang, Percy and Dan Klein (2009). "Online EM for Unsupervised Models". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 611619.

Littman, Michael L. (1994). "Markov games as a framework for multi-agent reinforcement learning". In: *11th Proc. of ICML*. Morgan Kaufmann, pp. 157–163.

Man vs Machine II - Polaris vs Online Poker's Best. `http://www.poker-academy.com/man-machine/2008/`. Accessed: 10/10/2013.

Marion R. Reynolds, Jr. and Zachary G. Stoumbos (1999). "A CUSUM Chart for Monitoring a Proportion When Inspecting Continuously". In: *Journal of Quality Technology* 31.1, pp. 87–108.

Mealing, Richard and Jonathan L. Shapiro (2013). "Opponent Modelling by Sequence Prediction and Lookahead in Two-Player Games". In: *12th International Conference on Artificial Intelligence and Soft Computing*, pp. 385–396.

— (2015). "Convergence of Strategies in Simple Co-Adapting Games". In: *Foundations of Genetic Algorithms XIII.*

— (under review). "Opponent Modelling by Expectation–Maximisation and Sequence Prediction in Simplified Poker". In: *IEEE Transactions on Computational Intelligence and AI in Games.*

Milgrom, Paul and John Roberts (1990). "Rationalizability, Learning, and Equilibrium in Games with Strategic Complementarities". In: *Econometrica* 58, pp. 1255–1277.

Millington, Ian (2006). "Artificial Intelligence for Games". In: ed. by David H. Eberly. Morgan Kaufmann. Chap. Learning, pp. 583–590.

Miyasawa, Koichi (1961). *On the Convergence of the Learning Process in a 2 x 2 Non-Zero-Sum Two-Person Game.* Tech. rep. Princeton University Econometric Research Program.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518, pp. 529–533.

Moffat, Alistair (1990). "Implementing the PPM Data Compression Scheme". In: *IEEE Transactions on Communications* 38, pp. 1917–1921.

Monderer, Dov and Lloyd S. Shapley (1996). "Fictitious Play Property for Games with Identical Interests". In: *Journal of Economic Theory* 68.1, pp. 258–265.

Nash, John F. (1950). "Equilibrium points in n-person games". In: *Proceedings of the National Academy of Sciences of the United States of America.*

Ny, Jerome Le (2006). "On Some Extensions of Fictitious Play". 2006.

Othello match of the year. `https://skatgame.net/mburo/event.html`. Accessed: 10/10/2013.

Page, E. S. (1954). "Continuous Inspection Schemes". In: *Biometrika* 41, pp. 100–115.

Piccolo, E. and G. Squillero (2011). "Adaptive opponent modelling for the iterated prisoner's dilemma". In: *IEEE CEC*, pp. 836–841.

Ponsen, Marc, Marc Lanctot, and Steven de Jong (2010). "MCRNR: Fast Computing of Restricted Nash Responses by Means of Sampling". In: *Interactive Decision Theory and Game Theory.*

Ponsen, Marc et al. (2008). "Bayes-Relational Learning of Opponent Models from Incomplete Information in No-Limit Poker". In: *Proceedings of the 23rd National Conference on Artificial Intelligence.*

Powers, Rob and Y. Shoham (2005). "New criteria and a new algorithm for learning in multi-agent systems". In: *Advances in neural information processing systems* 17, pp. 1089–1096.

Risk, Nick Abou and Duane Szafron (2010). "Using Counterfactual Regret Minimization to Create Competitive Multiplayer Poker Agents". In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems.*

Robinson, J. (1951). "An Iterative Method of Solving a Game". In: *Annals of Mathematics* 54, pp. 296–301.

Rock Paper Scissors Programming Competition. `http://www.rpscontest.com/`. Accessed: 12/03/2014.

Ross, Gordon J. (2013). *Parametric and Nonparametric Sequential Change Detection in R: The cpm package.* Tech. rep. Heilbronn Institute for Mathematical Research University of Bristol.

Ross, Gordon J., Dimitris K. Tasoulis, and Niall M. Adams (2013). "Sequential Monitoring of a Bernoulli Sequence when the Pre-Change Parameter is Unknown". In: *Comput. Stat.* 28, pp. 463–479.

Rubin, Jonathan and Ian Watson (2010). "Similarity-Based Retrieval and Solution Re-use Policies in the Game of Texas Hold'em". In: *Proceedings of the 18th International Conference on Case-Based Reasoning Research and Development.*

— (2011). "Computer poker: A review". In: *AI* 175.5–6, pp. 958–987. DOI: `10.1016/j.artint.2010.12.005`.

Sandholm, Tuomas (2010). "The State of Solving Large Incomplete-Information Games, and Application to Poker". In: *AI Magazine* 31.4, pp. 13–32.

Sato, Masa-Aki and Shin Ishii (2000). "On-line EM Algorithm for the Normalized Gaussian Network". In: *Neural Computation* 12.2, pp. 407–432. ISSN: 0899-7667.

Scerri, P. et al. (2010). "Towards an Understanding of the Impact of Autonomous Path Planning on Victim Search in USAR". In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on.*

Scrabble Showdown: Quackle vs. David Boys - Top 10 Man-vs.-Machine Moments - TIME. `http://content.time.com/time/specials/packages/article/0,28804,2049187_2049195_2049083,00.html`. Accessed: 10/10/2013.

Shapley, L.S. (1963). "Some Topics in Two-Person Games". In: *Advances in Game Theory* 3, pp. 1–28.

Shi, Jiefu and Michael L. Littman (2000). "Abstraction Methods for Game Theoretic Poker". In: *Revised Papers from the 2nd International Conference on Computers and Games.*

Singh, Satinder, Michael Kearns, and Yishay Mansour (2000). "Nash Convergence of Gradient Dynamics in General-Sum Games". In: *Sixteenth Conference on Uncertainty in Artificial Intelligence.*

Smyrnakis, Michalis and David S. Leslie (2010). "Dynamic Opponent Modelling in Fictitious Play". In: *The Computer Journal* 53, pp. 1344–1359.

— (2011). "Adaptive Forgetting Factor Fictitious Play". In: *ArXiv e-prints.* arXiv:1112.2315.

Southey, Finnegan et al. (2005). "Bayes' Bluff: Opponent Modelling in Poker". In: *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence.*

Stengel, Bernhard Von (1996). "Efficient Computation of Behavior Strategies". In: *Games and Economic Behavior* 14, pp. 220–246.

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning).* Ed. by T. Dietterich. The MIT Press.

Szepesvári, Cs (1998). *The Asymptotic Convergence–Rate of Q–learning.*

Tesauro, Gerald (1992). "Practical Issues in Temporal Difference Learning". In: *Machine Learning.*

— (1994). "TD-Gammon, a self-teaching backgammon program, achieves master-level play". In: *Neural Computation* 6, pp. 215–219.

— (1995). "Temporal difference learning and TD-Gammon". In: *Communications of the ACM* 38, pp. 58–68.

— (1998). "Comments on Co-Evolution in the Successful Learning of Backgammon Strategy". In: *Machine Learning* 32, pp. 241–243.

— (2002). "Programming backgammon using self-teaching neural nets". In: *Artificial Intelligence* 134, pp. 181–199.

Turner, Ryan, Yunus Saatci, and Carl Edward Rasmussen (2009). "Adaptive Sequential Bayesian Change Point Detection". In: *NIPS Temporal Segmentation Workshop.* URL: `http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.222.4513`.

Watkins, C. J. C. H. (1989). "Learning from delayed rewards". PhD thesis. Cambridge.

Wilson, Robert C., Matthew R. Nassar, and Joshua I. Gold (2010). "Bayesian Online Learning of the Hazard Rate in Change-Point Problems". In: *Neural Comput.* 22, pp. 2452–2476.

Wood, Frank et al. (2009). "A Stochastic Memoizer for Sequence Data". In: *ICML 26.*

Zhang, Chongjie and Victor Lesser (2010). "Multi-Agent Learning with Policy Prediction". In: *Proceedings of the 24th AAAI Conference on Artificial Intelligence.*

Zinkevich, Martin (2004). "Theoretical Guarantees for Algorithms in Multi-Agent Settings". PhD thesis. Carnegie Mellon University.

— (2005). "Response Regret". In: *AAAI.*

Zinkevich, Martin et al. (2008). "Regret Minimization in Games with Incomplete Information". In: *Advances in Neural Information Processing Systems 20.* MIT Press.

Appendices

Appendix A

Expectation-Maximisation

A.1 Categorical Distribution

Consider a categorical distribution with parameters $\vec{\mu} = (\mu_1, \mu_2, \ldots, \mu_D)$, where μ_i is the probability of sampling category i, $\sum_{i=1}^{D} \mu_i = 1$, $0 \leq \mu_i \leq 1$ for all $i \in \{1, 2, \ldots, D\}$. A categorical variable drawn from this distribution can be represented as a 1-of-D encoded vector $\vec{x} = (x_1, x_2, \ldots, x_D)$, where one component is equal to 1 and the rest are equal to 0. The probability of sampling $\vec{x}$ given $\vec{\mu}$ is

$$\Pr(\vec{x}|\vec{\mu}) = \prod_{i=1}^{D} \mu_i^{x_i}. \tag{A.1}$$

Suppose there is a data set X of N samples drawn from this categorical distribution $X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$. The likelihood of $\vec{\mu}$ given X is

$$L(\vec{\mu}; X) = \Pr(X|\vec{\mu}) = \prod_{n=1}^{N} \Pr(\vec{x}_n|\vec{\mu}) = \prod_{n=1}^{N} \prod_{i=1}^{D} \mu_i^{x_{ni}}. \tag{A.2}$$

The log-likelihood of $\vec{\mu}$ given X is

$$\ln L(\vec{\mu}; X) = \ln \Pr(X|\vec{\mu}) = \ln \prod_{n=1}^{N} \prod_{i=1}^{D} \mu_i^{x_{ni}} = \sum_{n=1}^{N} \sum_{i=1}^{D} x_{ni} \ln \mu_i. \tag{A.3}$$

Using a Lagrange multiplier λ to create a new function, which takes into account the constraint $\sum_{i=1}^{D} \mu_i = 1$, gives

$$G(\vec{\mu}, \lambda; X) = \Pr(X|\vec{\mu}, \lambda) = \left[\sum_{n=1}^{N} \sum_{i=1}^{D} x_{ni} \ln \mu_i \right] - \lambda \left[\left(\sum_{i=1}^{D} \mu_i \right) - 1 \right]. \tag{A.4}$$

Taking the partial derivatives of this function firstly with respect to a particular probability μ_c and secondly with respect to the Lagrange multiplier λ gives

$$\begin{aligned} \frac{\partial \Pr(X|\vec{\mu}, \lambda)}{\partial \mu_c} &= \frac{\partial}{\partial \mu_c} \left(\left[\sum_{n=1}^{N} \sum_{i=1}^{D} x_{ni} \ln \mu_i \right] - \lambda \left[\left(\sum_{i=1}^{D} \mu_i \right) - 1 \right] \right) \\ &= \left[\sum_{n=1}^{N} \frac{x_{nc}}{\mu_c} \right] - \lambda \end{aligned} \tag{A.5}$$

and

$$\begin{aligned} \frac{\partial \Pr(X|\vec{\mu}, \lambda)}{\partial \lambda} &= \frac{\partial}{\partial \lambda} \left(\left[\sum_{n=1}^{N} \sum_{i=1}^{D} x_{ni} \ln \mu_i \right] - \lambda \left[\left(\sum_{i=1}^{D} \mu_i \right) - 1 \right] \right) \\ &= 1 - \sum_{i=1}^{D} \mu_i. \end{aligned} \tag{A.6}$$

To find the maximising parameters, set the derivatives equal to zero, which gives

$$\lambda = \frac{1}{\mu_c} \sum_{n=1}^{N} x_{nc} \tag{A.7}$$

and

$$\sum_{i=1}^{D} \mu_i = 1. \tag{A.8}$$

With some manipulations, an expression for μ_c can be found as follows

$$\begin{aligned}
\lambda\mu_c &= \sum_{n=1}^{N} x_{nc} \\
\sum_{i=1}^{D} \lambda\mu_i &= \sum_{i=1}^{D}\sum_{n=1}^{N} x_{ni} \\
\lambda\sum_{i=1}^{D} \mu_i &= \sum_{i=1}^{D}\sum_{n=1}^{N} x_{ni} \\
\lambda &= \sum_{i=1}^{D}\sum_{n=1}^{N} x_{ni} \\
\mu_c &= \frac{1}{\lambda}\sum_{n=1}^{N} x_{nc} \\
\mu_c &= \frac{\sum_{n=1}^{N} x_{nc}}{\sum_{i=1}^{D}\sum_{n=1}^{N} x_{ni}}. \qquad \text{(A.9)}
\end{aligned}$$

To maximise the likelihood, set the probability of sampling category c from the categorical distribution as the number of times category c is sampled divided by the number of times any category is sampled.

A.2 Mixture of Categorical Distributions

Consider a mixture of K categorical distributions with parameters $\vec{\mu} = (\vec{\mu}_1, \vec{\mu}_2, \ldots, \vec{\mu}_K)$, where each $\vec{\mu}_k$ is defined in the same way as in Section A.1 and $\vec{\pi} = (\pi_1, \pi_2, \ldots, \pi_K)$, where π_k is the probability of sampling categorical distribution k, $\sum_{k=1}^{K} \pi_k = 1$, $0 \le \pi_k \le 1$ for all $k \in \{1, 2, \ldots, K\}$. The probability of sampling $\vec{x}$ given $\vec{\mu}$ and $\vec{\pi}$ is

$$\Pr(\vec{x}|\vec{\mu}, \vec{\pi}) = \sum_{k=1}^{K} \pi_k \Pr(\vec{x}|\vec{\mu}_k). \qquad \text{(A.10)}$$

Suppose that there is a data set X of N samples drawn from this mixture $X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$. The likelihood of $\vec{\mu}$ and $\vec{\pi}$ given X is

$$L(\vec{\mu}, \vec{\pi}; X) = \Pr(X|\vec{\mu}, \vec{\pi}) = \prod_{n=1}^{N}\sum_{k=1}^{K} \pi_k \Pr(\vec{x}_n|\vec{\mu}_k). \qquad \text{(A.11)}$$

The log-likelihood of $\vec{\mu}$ and $\vec{\pi}$ given X is

$$\ln L(\vec{\mu}, \vec{\pi}; X) = \ln \Pr(X|\vec{\mu}, \vec{\pi}) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \Pr(\vec{x}_n|\vec{\mu}_k). \tag{A.12}$$

Since a summation is inside the logarithm, the maximum likelihood does not have a closed-form solution. The next step is to derive the expectation-maximisation algorithm for maximising this likelihood function. For each instance of $\vec{x}$ introduce an explicit latent variable, which is a 1-of-K encoded vector $\vec{z} = (z_1, z_2, \ldots, z_K)$, where one component is equal to 1 and the rest are equal to 0, its value indicates which categorical distribution generated $\vec{x}$. The probability of $\vec{x}$ and $\vec{z}$ given $\vec{\mu}$ and $\vec{\pi}$ is

$$\Pr(\vec{x}, \vec{z}|\vec{\mu}, \vec{\pi}) = \prod_{k=1}^{K} \pi_k^{z_k} \Pr(\vec{x}|\vec{\mu}_k)^{z_k}. \tag{A.13}$$

The likelihood of $\vec{\mu}$ and $\vec{\pi}$ given X and Z is

$$\begin{aligned} L(\vec{\mu}, \vec{\pi}; X, Z) = \Pr(X, Z|\vec{\mu}, \vec{\pi}) &= \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \Pr(\vec{x}_n|\vec{\mu}_k)^{z_{nk}} \\ &= \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \left(\prod_{i=1}^{D} \mu_{ki}^{x_{ni}} \right)^{z_{nk}}. \end{aligned} \tag{A.14}$$

The log-likelihood of $\vec{\mu}$ and $\vec{\pi}$ given X and Z is

$$\begin{aligned} \ln L(\vec{\mu}, \vec{\pi}; X, Z) = \ln \Pr(X, Z|\vec{\mu}, \vec{\pi}) &= \ln \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \left(\prod_{i=1}^{D} \mu_{ki}^{x_{ni}} \right)^{z_{nk}} \\ &= \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \left(\ln \pi_k + \sum_{i=1}^{D} x_{ni} \ln \mu_{ki} \right). \end{aligned} \tag{A.15}$$

Taking the expected value with respect to the posterior distribution of Z gives

$$\mathbb{E}_Z[\ln \Pr(X, Z|\vec{\mu}, \vec{\pi})] = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \left(\ln \pi_k + \sum_{i=1}^{D} x_{ni} \ln \mu_{ki} \right) \tag{A.16}$$

where $\gamma(z_{nk}) = \mathbb{E}[z_{nk}]$ is the posterior probability or responsibility of categorical distribution k for sample $\vec{x}_n$. This is evaluated in the E-step as

$$\begin{aligned}\gamma(z_{nk}) = \mathbb{E}[z_{nk}] &= \frac{\sum_{\vec{z}_n} z_{nk} \prod_{k'} [\pi_{k'} \Pr(\vec{x}_n|\vec{\mu}_{k'})]^{z_{nk'}}}{\sum_{\vec{z}_n} \prod_j [\pi_j \Pr(\vec{x}_n|\vec{\mu}_j)]^{z_{nj}}} \\ &= \frac{\pi_k \Pr(\vec{x}_n|\vec{\mu}_k)}{\sum_{j=1}^{K} \pi_j \Pr(\vec{x}_n|\vec{\mu}_j)}. \qquad \text{(A.17)}\end{aligned}$$

Using a Lagrange multiplier $\vec{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_K)$ to create a new function, which takes into account the constraint $\sum_{i=1}^{D} \mu_{ki} = 1$ for all $k \in \{1, 2, \ldots, K\}$, gives

$$\begin{aligned}G(\vec{\mu}, \vec{\pi}, \vec{\lambda}; X, Z) &= \mathbb{E}_Z[\ln \Pr(X, Z|\vec{\mu}, \vec{\pi}, \vec{\lambda})] \\ &= \left[\sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \left(\ln \pi_k + \sum_{i=1}^{D} x_{ni} \ln \mu_{ki}\right)\right] \\ &\quad - \left[\sum_{k=1}^{K} \lambda_k \left[\left(\sum_{i=1}^{D} \mu_{ki}\right) - 1\right]\right]. \qquad \text{(A.18)}\end{aligned}$$

Taking the partial derivatives of this function firstly with respect to a particular probability μ_{dc} and secondly with respect to a particular Lagrange multiplier component λ_d gives

$$\begin{aligned}\frac{\partial \mathbb{E}_Z[\ln \Pr(X, Z|\vec{\mu}, \vec{\pi}, \vec{\lambda})]}{\partial \mu_{dc}} &= \frac{\partial}{\partial \mu_{dc}} \left(\left[\sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \left(\ln \pi_k + \sum_{i=1}^{D} x_{ni} \ln \mu_{ki}\right)\right]\right. \\ &\quad \left. - \left[\sum_{k=1}^{K} \lambda_k \left[\left(\sum_{i=1}^{D} \mu_{ki}\right) - 1\right]\right]\right) = \left[\sum_{n=1}^{N} \gamma(z_{nd}) \frac{x_{nc}}{\mu_{dc}}\right] - \lambda_d \\ &\qquad \text{(A.19)}\end{aligned}$$

and

$$\begin{aligned}\frac{\partial \mathbb{E}_Z[\ln \Pr(X, Z|\vec{\mu}, \vec{\pi}, \vec{\lambda})]}{\partial \lambda_d} &= \frac{\partial}{\partial \lambda_d} \left(\left[\sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \left(\ln \pi_k + \sum_{i=1}^{D} x_{ni} \ln \mu_{ki}\right)\right]\right. \\ &\quad \left. - \left[\sum_{k=1}^{K} \lambda_k \left[\left(\sum_{i=1}^{D} \mu_{ki}\right) - 1\right]\right]\right) = 1 - \sum_{i=1}^{D} \mu_{di}. \qquad \text{(A.20)}\end{aligned}$$

To find the maximising parameters, set the derivatives equal to zero, which gives

$$\lambda_d = \frac{1}{\mu_{dc}} \sum_{n=1}^{N} \gamma(z_{nd}) x_{nc} \tag{A.21}$$

and

$$\sum_{i=1}^{D} \mu_{di} = 1. \tag{A.22}$$

With some manipulations an expression for μ_{dc} can be found as follows

$$\begin{aligned}
\lambda_d \mu_{dc} &= \sum_{n=1}^{N} \gamma(z_{nd}) x_{nc} \\
\sum_{i=1}^{D} \lambda_d \mu_{di} &= \sum_{i=1}^{D} \sum_{n=1}^{N} \gamma(z_{nd}) x_{ni} \\
\lambda_d \sum_{i=1}^{D} \mu_{di} &= \sum_{i=1}^{D} \sum_{n=1}^{N} \gamma(z_{nd}) x_{ni} \\
\lambda_d &= \sum_{i=1}^{D} \sum_{n=1}^{N} \gamma(z_{nd}) x_{ni} \\
\mu_{dc} &= \frac{1}{\lambda_d} \sum_{n=1}^{N} \gamma(z_{nd}) x_{nc} \\
\mu_{dc} &= \frac{\sum_{n=1}^{N} \gamma(z_{nd}) x_{nc}}{\sum_{i=1}^{D} \sum_{n=1}^{N} \gamma(z_{nd}) x_{ni}}. \qquad \text{(A.23)}
\end{aligned}$$

To maximise the likelihood, set the probability of sampling category c from the categorical distribution d as the sum over all the samples of the responsibility of d to each sample multiplied by the value of that sample (1 if it was category c and 0 otherwise), divided by the sum of this calculation for all categories.

Appendix B

Generating Payoffs for a Nash Equilibrium in a Two-Player Two-Action Game

The goal is to generate payoff matrices for each player that describe a two-player, two-action, normal-form game such that the Nash equilibrium of the game is at a given point (p*,q*). Let

$$
\begin{aligned}
p &= \text{probability row player 1 plays action } r_1,\\
(1-p) &= \text{probability row player 1 plays action } r_2,\\
q &= \text{probability column player 2 plays action } c_1,\\
(1-q) &= \text{probability column player 2 plays action } c_2,
\end{aligned}
$$

$$
\begin{aligned}
p* &= \text{Nash equilibrium probability row player 1 plays action } r_1,\\
(1-p*) &= \text{Nash equilibrium probability row player 1 plays action } r_2,\\
q* &= \text{Nash equilibrium probability column player 2 plays action } c_1, \text{ and}\\
(1-q) &= \text{Nash equilibrium probability column player 2 plays action } c_2
\end{aligned}
$$

B.1 Payoffs for a Nash Equilibrium at a Centre Point

Table B.1: Payoffs for a two-player, two-action, normal-form game with a Nash equilibrium at (p*,q*). The gradients orbit the Nash equilibrium.

	c_1	c_2
r_1	$\frac{2}{q*}$ - 3,-$\frac{2}{p*}$ + 3	-1,1
r_2	-1,1	1,-1

The expected payoff to player 1 is

$$\begin{aligned} V_1 &= pq\left(\frac{2}{q*} - 3\right) + p(1-q)(-1) + (1-p)q(-1) + (1-p)(1-q)(1) \\ &= pq\left(\frac{2}{q*} - 3\right) - p(1-q) - (1-p)q + (1-p)(1-q) \\ &= pq\left(\frac{2}{q*}\right) - 2p - 2q + 1. \end{aligned}$$

The gradient of V_1 with respect to p is

$$\frac{\partial V_1}{\partial p} = q\left(\frac{2}{q*}\right) - 2.$$

Thus, if $q = q*$ then $\frac{\partial V_1}{\partial p} = 0$.

The expected payoff to player 2 is

$$\begin{aligned} V_2 &= pq\left(-\frac{2}{p*} + 3\right) + p(1-q)(1) + (1-p)q(1) + (1-p)(1-q)(-1) \\ &= pq\left(-\frac{2}{p*} + 3\right) + p(1-q) + (1-p)q - (1-p)(1-q) \\ &= pq\left(-\frac{2}{p*}\right) + 2p + 2q - 1. \end{aligned}$$

The gradient of V_2 with respect to q is

$$\frac{\partial V_2}{\partial q} = p\left(-\frac{2}{p*}\right) + 2.$$

Thus, if $p = p*$ then $\frac{\partial V_2}{\partial q} = 0$.

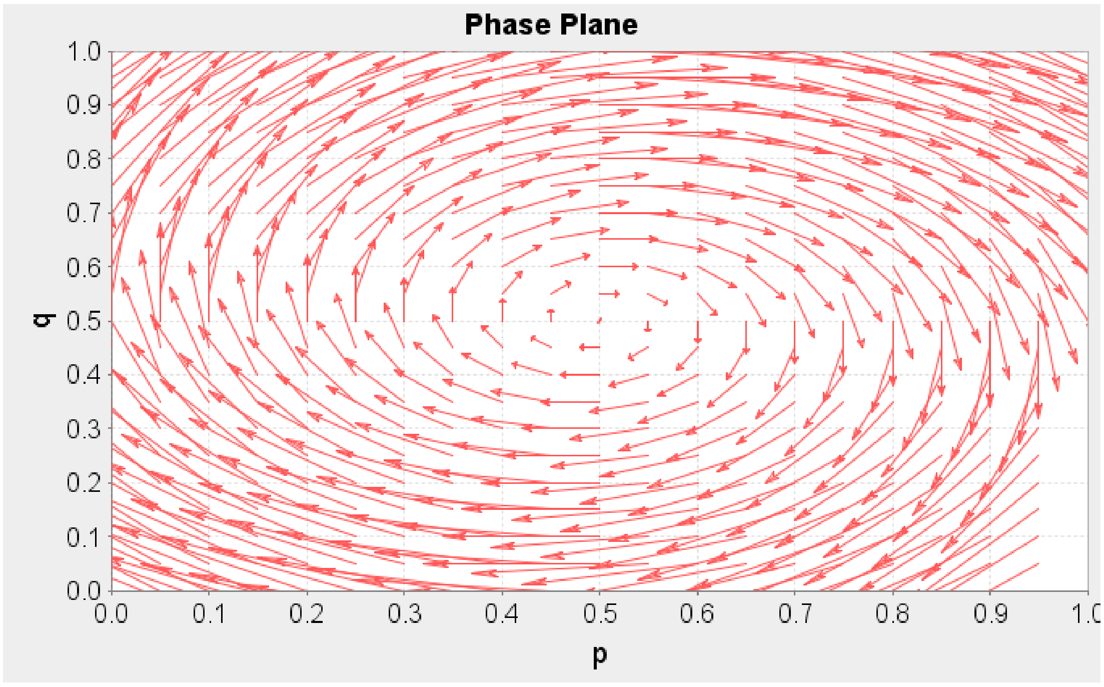


Figure B.1: Gradients around a centre Nash equilibrium at (p* = 0.5, q* = 0.5).

B.2 Payoffs for a Nash Equilibrium at a Saddle Point

Table B.2: Payoffs for a two-player, two-action normal-form game with a Nash equilibrium at (p*,q*). The gradients form a saddle at the Nash equilibrium.

	c_1	c_2
r_1	1 - q*,1 - p*	-q*,0
r_2	0,-p*	0,0

The expected payoff to player 1 is

$$\begin{aligned}
V_1 &= pq\,(1-q*) + p(1-q)(-q*) + (1-p)q(0) + (1-p)(1-q)(0) \\
&= pq\,(1-q*) + p(1-q)(-q*) \\
&= pq - pq*.
\end{aligned}$$

The gradient of V_1 with respect to p is

$$\frac{\partial V_1}{\partial p} = q - q*.$$

Thus, if $q = q*$ then $q - q* = 0$.

The expected payoff to player 2 is

$$\begin{aligned}
V_2 &= pq\,(1-p*) + p(1-q)(0) + (1-p)q(-p*) + (1-p)(1-q)(0) \\
&= pq\,(1-p*) + (1-p)q(-p*) \\
&= pq - p*q.
\end{aligned}$$

The gradient of V_2 with respect to q is

$$\frac{\partial V_2}{\partial q} = p - p*.$$

Thus, if $p = p*$ then $\dfrac{\partial V_2}{\partial q} = 0$.

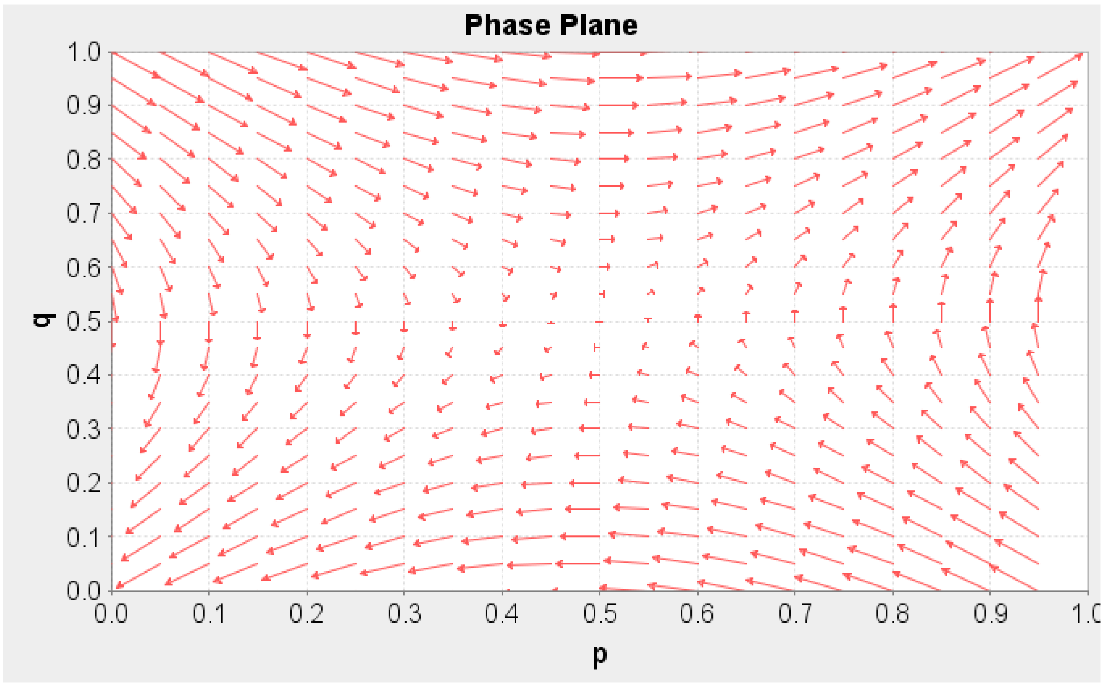


Figure B.2: Gradients around a saddle Nash equilibrium at ($p^* = 0.5$, $q^* = 0.5$).

Appendix C

Simplex Sampling and Simplex Random Walk

C.1 Sampling a Random Point on the $K-1$-Dimensional Simplex

The goal is to sample a random point from the $K-1$-dimensional simplex. This is equivalent to sampling from a Dirichlet distribution with a K-dimensional concentration parameter of ones. The Dirichlet distribution is defined as

$$\mathrm{Dir}(x_1, x_2, \ldots, x_K; \alpha_1, \alpha_2, \ldots, \alpha_K) = \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \prod_{i=1}^{K} x_i^{\alpha_i - 1} \tag{C.1}$$

where $(x_1, x_2, \ldots, x_K)$ is a categorical distribution such that $0 \leq x_i \leq 1$ for $i \in \{1, 2, \ldots, K\}$, $\sum_{i=1}^{K} x_i = 1$, and $(\alpha_1, \alpha_2, \ldots, \alpha_K)$ is the concentration parameter such that $\alpha_i > 0$ for $i \in \{1, 2, \ldots, K\}$. It has been proven by Devroye, 1986 with a concentration parameter $(\alpha_1, \alpha_2, \ldots, \alpha_K)$, a scale parameter θ, and samples of K random variables $(X_1, X_2, \ldots, X_K)$ from gamma distributions using these parameters such that $X_i \sim \Gamma(\alpha_i, \theta)$ then

$$(Y_1, Y_2, \ldots, Y_K) \sim \mathrm{Dir}(\alpha_1, \alpha_2, \ldots, \alpha_K) \text{ where } Y_i = \frac{X_i}{\sum_{j=1}^{K} X_j} \text{ for } 1 \leq i \leq K. \tag{C.2}$$

Here the gamma distribution is defined as

$$\Gamma(x;\alpha,\theta)=\frac{x^{\alpha-1}e^{-\frac{x}{\theta}}}{\theta^{\alpha}\Gamma(\alpha)} \tag{C.3}$$

where $x>0$, $\alpha>0$ is the shape parameter, and $\theta>0$ is the scale parameter.

The scale parameter, θ, shared by the Gamma distributed random variables in Equation C.2 can be arbitrarily set to any value. To see this, recall that multiplying a Gamma distributed random variable by a strictly positive constant produces another Gamma distributed random variable. Specifically, if $X\sim\Gamma(\alpha,\theta)$, then $cX\sim\Gamma(\alpha,c\theta)$. Thus, if $X\sim\Gamma(\alpha,\theta)$ then $\theta^{-1}X\sim\Gamma(\alpha,1)$. This means that in Equation C.2 each random variable $X_i\sim\Gamma(\alpha_i,\theta)$ can be represented as $\theta X_i'$ where $X_i'\sim\Gamma(\alpha_i,1)$. Therefore, $Y_i=X_i/\sum_{j=1}^{K}X_j=\theta X_i'/\sum_{j=1}^{K}\theta X_j'=X_i'/\sum_{j=1}^{K}X_j'$, which shows that the scale parameter is factored out in the normalisation.

To prove that, in Equation C.2, $(Y_1,Y_2,\ldots,Y_K)$ has a Dirichlet distribution, the proof of Devroye, 1986 is followed with help from Frigyik, Kapila, and Gupta, 2010. The change-of-variables formula is used to show that the density of $(Y_1,Y_2,\ldots,Y_K)$ is the same as the density of a Dirichlet distribution. The original random variables are $(X_1,X_2,\ldots,X_K)$ where $X_i\sim\Gamma(\alpha_i,\theta)$, and the new random variables are $(Y_1,Y_2,\ldots,Y_{K-1},Z)$, where $Z=\sum_{i=1}^{K}X_i$, $Y_i=X_i/\sum_{j=1}^{K}X_j=X_i/Z$, and so $X_i=ZY_i$. Since the scale parameter, θ, can be set arbitrarily, it is set to one, $\theta=1$, to simplify the proof. These variables are related using the invertible transformation $T:\mathbb{R}^K\to\mathbb{R}^K$, a vector-valued function, as follows

$$\begin{aligned}(X_1,X_2,\ldots,X_K) &= T(Y_1,Y_2,\ldots,Y_{K-1},Z)\\ &= \left(ZY_1,ZY_2,\ldots,ZY_{K-1},Z\left(1-\sum_{i=1}^{K-1}Y_i\right)\right).\end{aligned} \tag{C.4}$$

The transformation T is given by K real-valued component functions

$$\begin{aligned}
T_1(Y_1, Y_2, \ldots, Y_{K-1}, Z) &= ZY_1, \\
T_2(Y_1, Y_2, \ldots, Y_{K-1}, Z) &= ZY_2, \\
&\vdots \\
T_{K-1}(Y_1, Y_2, \ldots, Y_{K-1}, Z) &= ZY_{K-1}, \\
T_K(Y_1, Y_2, \ldots, Y_{K-1}, Z) &= Z\left(1 - \sum_{i=1}^{K-1} Y_i\right).
\end{aligned}$$

The Jacobian matrix (matrix of all first-order partial derivatives) of T, $J(T)$, is

$$\begin{aligned}
J(T) &= \begin{pmatrix}
\frac{\partial T_1}{\partial Y_1} & \frac{\partial T_1}{\partial Y_2} & \cdots & \frac{\partial T_1}{\partial Y_{K-1}} & \frac{\partial T_1}{\partial Z} \\
\frac{\partial T_2}{\partial Y_1} & \frac{\partial T_2}{\partial Y_2} & \cdots & \frac{\partial T_2}{\partial Y_{K-1}} & \frac{\partial T_2}{\partial Z} \\
\vdots & \ldots & \ldots & \ldots & \vdots \\
\frac{\partial T_{K-1}}{\partial Y_1} & \frac{\partial T_{K-1}}{\partial Y_2} & \cdots & \frac{\partial T_{K-1}}{\partial Y_{K-1}} & \frac{\partial T_{K-1}}{\partial Z} \\
\frac{\partial T_K}{\partial Y_1} & \frac{\partial T_K}{\partial Y_2} & \cdots & \frac{\partial T_K}{\partial Y_{K-1}} & \frac{\partial T_K}{\partial Z}
\end{pmatrix} \\
&= \begin{pmatrix}
Z & 0 & \ldots & 0 & Y_1 \\
0 & Z & \ldots & 0 & Y_2 \\
\vdots & \ldots & \ldots & \ldots & \vdots \\
0 & 0 & \ldots & Z & Y_{K-1} \\
-Z & -Z & \ldots & -Z & 1 - \sum_{i=1}^{K-1} Y_i
\end{pmatrix}. \qquad \text{(C.5)}
\end{aligned}$$

To calculate the determinant, the fact that adding rows does not change the determinant can be used. Adding the first $K-1$ rows to the last row gives

$$\begin{pmatrix}
Z & 0 & \ldots & 0 & Y_1 \\
0 & Z & \ldots & 0 & Y_2 \\
\vdots & \ldots & \ldots & \ldots & \vdots \\
0 & 0 & \ldots & Z & Y_{K-1} \\
0 & 0 & \ldots & 0 & 1
\end{pmatrix},$$

which is an upper diagonal matrix (elements below the diagonal are zero). The determinant of an upper diagonal matrix is the product of the diagonal elements. Thus, the determinant of $J(T)$ is $\det(J(T)) = Z^{K-1}$. The change-of-variables formula tells us that the density of $(Y_1, Y_2, \ldots, Y_{K-1}, Z)$ is $f = g \circ T \times |\det(T)|$.

Here g is the joint density of the original (independent) random variables

$$g(x_1, x_2, \ldots, x_K; \alpha_1, \alpha_2, \ldots, \alpha_K) = \prod_{i=1}^{K} x_i^{\alpha_i - 1} \frac{e^{-x_i}}{\Gamma(\alpha_i)}. \tag{C.6}$$

Substituting in the components of the change-of-variables formula gives

$$\begin{aligned}
& f(y_1, y_2, \ldots, y_{K-1}, z) \\
= \; & \prod_{i=1}^{K-1} (zy_i)^{\alpha_i - 1} \frac{e^{-(zy_i)}}{\Gamma(\alpha_i)} \left[\left(z \left(1 - \sum_{i=1}^{K-1} y_i \right) \right)^{\alpha_K - 1} \frac{e^{-(z(1-\sum_{i=1}^{K-1} y_i))}}{\Gamma(\alpha_K)} \right] z^{K-1} \\
= \; & \frac{\left(\prod_{i=1}^{K-1} y_i^{\alpha_i - 1} \right) \left(1 - \sum_{i=1}^{K-1} y_i \right)^{\alpha_K - 1}}{\prod_{i=1}^{K} \Gamma(\alpha_i)} z^{(\sum_{i=1}^{K} \alpha_i) - 1} e^{-z}.
\end{aligned} \tag{C.7}$$

Finally, integrating over z, the marginal distribution of $(Y_1, Y_2, \ldots, Y_{K-1})$ is

$$\begin{aligned}
& f(y_1, y_2, \ldots, y_{K-1}) \\
= \; & \int_0^\infty f(y_1, y_2, \ldots, y_{K-1}, z) dz \\
= \; & \frac{\left(\prod_{i=1}^{K-1} y_i^{\alpha_i - 1} \right) \left(1 - \sum_{i=1}^{K-1} y_i \right)^{\alpha_K - 1}}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \int_0^\infty z^{(\sum_{i=1}^{K} \alpha_i) - 1} e^{-z} dz \\
= \; & \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \left(\prod_{i=1}^{K-1} y_i^{\alpha_i - 1} \right) \left(1 - \sum_{i=1}^{K-1} y_i \right)^{\alpha_K - 1}
\end{aligned} \tag{C.8}$$

which is by definition a Dirichlet distribution.

C.2 Performing a Random Walk on the $K-1$-Dimensional Simplex

The goal is to perform a random walk on the $K-1$-dimensional simplex. To do this, a Metropolis-Hastings algorithm is used to produce a sequence of samples, which randomly walk over it, and whose stationary distribution is uniform over it. This method is explained in Fernandes and Atchley, 2008 and Fernandes, 2008.

C.2.1 Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm is a Markov chain Monte Carlo method for generating a sequence of random samples from a probability distribution $p(x)$. It requires you to be able to evaluate some probability distribution $f(x)$ that is proportional to $p(x)$, i.e. $f(x) \propto p(x)$. It also requires a proposal distribution $q(x)$ to move around the sample space. Each new sample, x_{new}, is generated from the proposal distribution, which usually depends on the old sample, x_{old}, i.e. the proposal distribution is usually $q(x_{\text{new}}|x_{\text{old}})$. The new sample is accepted with a probability of $\min(1, f(x_{\text{new}})q(x_{\text{old}}|x_{\text{new}})/f(x_{\text{old}})q(x_{\text{new}}|x_{\text{old}}))$. In this way, the current sample randomly moves (walks) within the sample space over time, sometimes moving to a new position depending on the old position, and sometimes staying in place. As more samples are generated, their distribution more closely approximates the desired distribution $p(x)$.

C.2.2 Application of the Metropolis-Hastings Algorithm

To perform a random walk on the $K-1$-dimensional simplex, first sample an initial point, which is a categorical distribution with K-categories, using the procedure outlined in Section C.1, i.e. sample $(X_1, X_2, \ldots, X_K)$ where $X_i \sim \Gamma(1,1)$ (unit exponential) for $i \in \{1, 2, \ldots, K\}$ and normalise them to produce $(Y_1, Y_2, \ldots, Y_K)$ where $Y_i = X_i / \sum_{j=1}^{K} X_j$ for $i \in \{1, 2, \ldots, K\}$. Instead of performing the random walk by changing the probabilities of the categorical distribution directly, instead change the unit-exponential random samples, which were normalised to give those probabilities, i.e. change X_i instead of Y_i. Following this, update each unit-exponential random sample at each time step using a Metropolis-Hastings algorithm. The sequence of updates to each value form a random walk. To get the updated categorical distribution at any time step, simply normalise the values of the unit-exponential random samples at that time step. However, be sure to keep the unnormalised values as these are what are being updating.

The algorithm updates a unit-exponential random sample, x_{old}, by performing a single Metropolis-Hastings step on it. The resulting new unit-exponential random sample, x_{new}, is based on the old value.

- The stationary (desired) distribution, $p(x)$, is unit-exponential $p(x) = e^{-x}$.
- The proportional distribution, $f(x)$, is $f(x) = p(x) = e^{-x}$.

- The proposal distribution, $q(x'|x)$, is

$$\begin{aligned} x' &\leftarrow xe^Z \text{ where } Z \sim \mathcal{N}(0,h) \\ &\leftarrow e^{\ln(x)}e^Z \\ &\leftarrow e^{\ln(x)+Z} \\ &\leftarrow e^{Z'} \text{ where } Z' \sim \mathcal{N}(\ln(x),h). \end{aligned} \tag{C.9}$$

  Thus, the probability distribution for x' is a log-normal distribution with parameters $\mu = \ln(x)$ and $\sigma = h$, which can be written as

$$q(x'|x) = \frac{1}{x'h\sqrt{2\pi}} e^{\frac{-(\ln(x')-\ln(x))^2}{2h^2}}. \tag{C.10}$$

- The Hastings ratio, r_h, is

$$r_h = \frac{q(x_{\text{old}}|x_{\text{new}})}{q(x_{\text{new}}|x_{\text{old}})} = \frac{e^{\frac{-(\ln(x_{\text{old}})-\ln(x_{\text{new}}))^2}{2h^2}}}{x_{\text{old}}h\sqrt{2\pi}} \frac{x_{\text{new}}h\sqrt{2\pi}}{e^{\frac{-(\ln(x_{\text{new}})-\ln(x_{\text{old}}))^2}{2h^2}}} = \frac{x_{\text{new}}}{x_{\text{old}}}. \tag{C.11}$$

- The Metropolis ratio, r_m, is

$$r_m = \frac{f(x_{\text{new}})}{f(x_{\text{old}})} = \frac{e^{-x_{\text{new}}}}{e^{-x_{\text{old}}}}. \tag{C.12}$$

- The acceptance probability distribution, $a(x_{\text{new}}|x_{\text{old}})$, is

$$a(x_{\text{new}}|x_{\text{old}}) = \min\left(1, \frac{f(x_{\text{new}})q(x_{\text{old}}|x_{\text{new}})}{f(x_{\text{old}})q(x_{\text{new}}|x_{\text{old}})}\right) = \min(1, r_h r_m). \tag{C.13}$$

Given these definitions, the algorithm is as follows:

1. Require a state x_{old}.
2. Calculate a new state $x_{\text{new}} \leftarrow x_{\text{old}}e^Z$ where $Z \sim \mathcal{N}(0,h)$.
3. Accept the update, $x_{\text{old}} \leftarrow x_{\text{new}}$, with probability $\min\left(1, \frac{e^{-x_{\text{new}}}x_{\text{new}}}{e^{-x_{\text{old}}}x_{\text{old}}}\right)$.

This algorithm is used to update each unit-exponential random variable at each time step, and is equivalent to Algorithm 15.

Appendix D

Fictitious Play Example Details

The details presented here were derived in Mealing and Shapiro, 2015. The dynamics in expectation of fictitious play, represented by equations 7.1 and 7.2, are solved for one cycle; a cycle is shown in Figure 7.1b. Each arm of the diamond shaped cycle is a period of time when one agent is playing a winning strategy, and the other is playing a losing strategy. At the end of the cycle the system is closer to the origin. The calculation works as follows. One calculates the time to traverse each of the four arms of the diamond shaped cycle, and the locations of each of its vertices. The calculation is slightly complicated by the first step of each arm, in which one of the strategies is updated by 0 instead of ± 1. In the next theorem, the calculation is shown.

Theorem D.0.1 *Starting the dynamical system at time t_0 with $y(t_0) = y_0$ and $x(t_0) = 0$, and assuming that $y(t_0)$ is the time average of a series of values, where each value is -1 or 1 (so that when it changes sign, it will go through the value 0 exactly), the time taken to traverse the cycle is*

$$T_1 = 4t_0 y_0 + 10, \tag{D.1}$$

and the value of y at the end of the cycle is,

$$y(t_0 + T_1) = \frac{t_0 y_0 + 4}{t_0(1 + 4y_0) + 10}. \tag{D.2}$$

Proof A solution to equations 7.1 and 7.2 will take the form,

$$a_i(t+\tau) = \frac{t a_i(t)}{t+\tau} \pm \begin{cases} \frac{\tau-1}{t+\tau} & \text{if opponent } a_{-i}(t) = 0, \\ \frac{\tau}{t+\tau} & \text{otherwise.} \end{cases} \tag{D.3}$$

Here a_i is either x or y, a_{-i} is the alternative, and the sign is positive if a_i is increasing or negative if a_i is decreasing. The time period, τ, is between changes of strategy. Traversing an arm of the cycle (or a quarter cycle), starts with $a_i = 0$ and finishes when $a_{-i} = 0$. The time taken, and values of a_i and a_{-i} after this time will be

$$\tau = ta_i + 1, \tag{D.4}$$

$$a_{-i} = \pm\frac{ta_i + 1}{t + ta_i + 1} \tag{D.5}$$

$$a_i = 0. \tag{D.6}$$

Here t is the time at the start of the traversal of this arm. To get the properties of the cycle, we just have to iterate this four times starting at time t_0, with $x(t_0) = 0$ and $y(t_0)$ positive, and alternate the roles of x and y. First, subtracting from y and adding to x until $y = 0$ (arm 1), then subtracting from y and x until $x = 0$ (arm 2), then adding to y and subtracting from x until $y = 0$ (arm 3), then adding to y and x until $x = 0$ which completes the cycle. Iterating Equations D.4 and D.5 four times gives the result.

In order to verify the claims made in Section 7.1, the following two results can be used.

Corollary D.0.2 *The period of the ith cycle is proportional to i, and the time after the ith cycle is $O(i^2)$.*

Proof Define T_i as the period of the ith cycle, t_i as the time after the ith cycle, and y_i as the value of y after the ith cycle. The recurrence relations implied by Equations D.1 and D.2 are,

$$T_i = 4t_{i-1}y_{i-1} + 10, \tag{D.7}$$

$$y(t_{i-1} + T_i) = y_i = \frac{t_{i-1}y_{i-1} + 4}{t_{i-1}(1 + 4y_{i-1}) + 10} \tag{D.8}$$

Using $t_{i-1} = t_{i-2} + T_{i-1}$ and the value for y_{i-1} from Equation D.8 yields the recursion relation

$$T_i = 4(t_{i-2}(1 + 4y_{i-2}) + 10)\frac{t_{i-2}y_{i-2} + 4}{t_{i-2}(1 + 4y_{i-2} + 10} = T_{i-1} + 16. \tag{D.9}$$

This is solved as $T_i = T_{i-1} + 16(i-1)$. From an asymptotic perspective, this proves the result. The time after i cycles is

$$\begin{aligned} t_i &= t_0 + \sum_{j=1}^{i} T_j = t_0 + iT_1 + 16\sum_{j=1}^{i-1} j = t_0 + iT_1 + 16\frac{i(i-1)}{2} \\ &= t_0 + i(T_1 - 8) + 8i^2 \text{ which is } O(i^2). \end{aligned} \tag{D.10}$$

A starting point consistent with the assumptions is $t_0 = 1$, $y_0 = 1$, and $x_0 = 0$. Thus, $T_1 = 4t_0y_0 + 10 = 14$, and

$$t_i = t_0 + i(T_1 - 8) + 8i^2 = 1 + 6i + 8i^2. \tag{D.11}$$

So, the cycle period grows like i, and the time between cycles grows like i^2.

Corollary D.0.3 *The system converges to* 0 *in inverse proportion to the number of cycles.*

Proof The task is to show that y decreases like $1/i$. We need to solve recursion relation D.8. It is helpful to see that t_i from Equation D.11 (where $t_0 = 1$, $y_0 = 1$, and $x_0 = 0$) can be factorised as $(1 + 2i)(1 + 4i)$. We solve recursion relation D.8 by ansatz, guessing that $y_i = 1/(1 + 2i)$. Due to the factorisation, $y_it_i = 1 + 4i$, which gives

$$\begin{aligned} y_{i+1} &= \frac{4i+5}{(1+2i)(1+4i) + 4(1+4i) + 10} = \frac{4i+5}{8i^2 + 22i + 15} \\ &= \frac{1}{2i+3} = \frac{1}{1+2(i+1)} \end{aligned} \tag{D.12}$$

So the ansatz works, and y shrinks per cycle like $\Theta(1/i)$.

Corollary D.0.4 *The system defined by recurrence relations 7.1 and 7.2 converges like inverse square-root of time.*

Proof According to Corollary D.0.3, the system gets closer to the fixed point inversely with the number of cycles, and due to Corollary D.0.3, the time to complete i cycles scales like i^2. Thus, in time t^2 the system gets closer to the fixed point by $1/t$, so in time t it gets closer to the fixed point by $1/\sqrt{t}$.