University of Alberta

USING STATE ESTIMATION FOR DYNAMIC AGENT MODELLING

by

Nolan Deans Carlisle Bard

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Spring 2008

Permission is hereby granted to the University of Alberta Libraries to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only. Where the thesis is converted to, or otherwise made available in digital form, the University of Alberta will advise potential users of the thesis of these terms.

©The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatsoever without the author's prior written permission.

To my family and friends,
you will always be the best part of my life.
Thank you for tolerating my idiosyncrasies over the years.

Abstract

Agent modelling is a challenging problem in many modern artificial intelligence applications. The agent modelling task is especially difficult when handling stochastic choices, deliberately hidden information, dynamic agents, and the need for fast learning. State estimation techniques, such as Kalman filtering and particle filtering, have addressed many of these challenges, but have received little attention in the agent modelling literature. This thesis explores the use of particle filtering for modelling dynamic opponents in Kuhn poker, a simplified version of Texas Hold'em poker. We demonstrate effective modelling both against static opponents as well as dynamic opponents, when the dynamics are known. We then examine an application of Rao-Blackwellized particle filtering for doing dual estimation, inferring both the opponent's state and a model of its dynamics. Finally, we examine the robustness of the approach to incorrect beliefs about the opponent and compare it to previous work on opponent modelling in Kuhn poker.

Acknowledgements

Throughout my years at the University of Alberta I have had the pleasure of working and having fun with many exceptional people. All of these people deserve thanks for making life at school a wonderful experience that I will always cherish.

First, I would like to thank my supervisor, **Michael Bowling**. When I started my master's degree I thought a supervisor would simply be a person that helped guide my research. Michael filled that role admirably and, moreover, he has become a friend. He has provided me with both the freedom and opportunity to grow as a scientist. His knowledge, insight, guidance, wit, and support have made the academic side of my degree very satisfying. Finally, Michael's kindness and friendship allowed me to know my supervisor more personally. I will remember our trips to the climbing wall and American Thanksgivings fondly. I also feel that I must thank Michael's wife **Shayna Bowling**. She, like Michael, exudes kindness and I appreciate her patience when I needed to borrow her husband.

Next, I would like to thank the members of the Computer Poker Research Group (CPRG). Several of these people require some special recognition. **Rob Holte** and **Bret Hoehn** for their insight and expertise in the domain of Kuhn poker. **Darse Billings** for your joviality and for helping me in the first summer of my master's program when I was working on card isomorphisms. **Michael Johanson** for helping me to create scripts for running my many jobs on the eureka cluster. **Neil Burch** for your coding and mathematical prowess that both inspired and educated beyond any of my expectations. Finally, the rest of the CPRG also deserves my thanks for being such a tight-knit and highly collaborative team. Thanks to each of **Jonathan Schaeffer, Duane Szafron, Martin Zinkevich, Michael Bowling, Morgan Kan, Josh Davidson, Carmelo Piccione, Andrew Albert, and John Hawkin**. Because of all of you, the Man-Machine poker competition was a great success that I was fortunate to be involved in.

The last few years of my life have also been driven by people and friendships. All of my friends had either a direct or indirect impact on this research. Between shaping who I am, keeping me sane, and providing me with lifelong memories, I owe you all more than I can ever repay. My friends, **Mark Lee** and **Dirk Henkemans**, were both strong influences in my interest in computing science. Although Dirk passed away in 2004 just before we finished our bachelor's degree, both Mark and Dirk have had a profound impact on shaping who I am today. **Daniel Lizotte** also deserves special thanks for being my "go-to guy" on topics ranging from statistics to Matlab to Unix (and for being amazing in general). **Brad Joyce** has thoroughly filled the ingenious-quirky-insomniac-Saskatchewanian niche that I never knew needed filling. His futon also provided frequent refuge from a long walk home after a night on Whyte Avenue. **Jessica Enright** deserves special thanks for making tasty food and keeping us all prepared for the Zombiepocalypse. I owe **Chris Rayner** gratitude for letting me leech off his determination for staying physically active despite the sedentary lifestyle of a computing scientist. Without him, I would have never discovered my love for judo. My friend and former roommate, **Andrew Butcher**, deserves recognition for consistently drawing me into spectacularly farcical debates. **Leah Hackman** gets a gold star for being the most exuberant person I know and for being a knitting ninja. **Morgan Kan** has always been as reliable as the tides and fun to antagonize with a game of Arkham Horror. **Curtis Onuczko** is one of the most predictably erratic people I know and we both suffered through the challenge of making a Blokus AI. I will always envy **Michael Johanson** for his ability to write and program faster than the laws of nature allow. Thanks to **Jeff Ryan** for being another person who gets excited about cartoons. **Adam White**, you are the world's kindest magnificent jerk. **Joel Jackson**, thank you for letting me call you a "yuppie cowboy" without getting visibly upset. **Jeff Siegel**, your rational voice is always appreciated – especially since you appear immune to Curtis' erratic behaviour. **Paul Berube**, in you I have found another person who hungers for skiing and snowboarding. To everyone in the Artificial Intelligence lab, thank you for your kindness and tolerance. Never forget our delicious bread. To everyone in the RLAI lab, thank you for letting me enjoy your company and your beer nights. I could keep going, but the orchestra is playing and the audience is getting rowdy. If I forgot you and you feel bad, just get in touch with me and I will extol your virtues.

Finally, I would like to thank my family. My mother, **Jeanne Byron**, and father, **Forrest Bard**, have always supported and encouraged me. You have shaped my life more than anyone else. My brother, **Ryan Bard**, has been like a brother to me: supporting me, encouraging me, and bailing me out of my self-inflicted train wrecks while still quarrelling with me just enough to remind me how similar we can be. To all of my family, thank you for always being there for me and supporting me. As quirky as we all are, I am fortunate to have you all as my family.

Table of Contents

List of Tables

List of Figures

List of Symbols

Undominated Kuhn Poker Parameters

Symbol	Definition
α	Player one's probability of betting in round one with the jack
β	Player one's probability of betting in round three with the queen
γ	Player one's probability of betting in round one with the king
η	Player two's probability of betting in round two with the queen following a bet from player one
ξ	Player two's probability of betting in round two with the jack following a pass from player one

Dynamics Model Parameters

Symbol	Definition
ρ	Switching motion model parameter specifying the probability of switching to a new strategy after a hand
σ	Drifting motion model parameter specifying the standard deviation of the Gaussian

Chapter 1

Introduction

The rise of applications requiring interaction between independent agents has made coping with the presence of other decision makers a key challenge for artificial intelligence. Agent modelling is one tool for addressing this challenge. We begin by introducing agent modelling, its importance, and the challenges one faces when modelling agents. Following this discussion, we present the thesis' contributions to the field of agent modelling and an outline of the thesis.

1.1 Agent Modelling

Most people have daily interaction with a computer agent. Whether they are paying their phone bill through the phone company's automated receptionist, searching the Internet, or simply riding an elevator, computer agents make decisions that impact peoples' lives. Given the capacity to reason about the behaviour of other agents, especially humans, these computer agents could assist people more effectively. Agent modelling research aims to give computer agents this capability.

Formally, an **agent** is any autonomous decision maker – human or computer. Agent modelling aims to enable computer agents to infer the beliefs, plans, or goals of other agents. More specifically, agent modelling is the problem of building a predictive model of another agent's future decisions from, possibly incomplete, observations of past behavior. With an accurate predictive model of the other agents, a computer agent can plan and execute a more effective response.

In addition to being beneficial to mundane day-to-day tasks, agent modelling can be applied to domains throughout the artificial intelligence (AI) community. Applications as diverse as assistive technologies, autonomous driving, and interactive entertainment all require or would benefit significantly from accurate models of other agents, artificial or human.

Many domains have features that make it difficult to model other agents. These features include:

- **Limited observations.** We usually need to model agents quickly. This mean that we need to build a model using a small number of observations.
- **Stochastic observations.** Other agents may make decisions stochastically (*i.e.* acting according to some probability distribution over possible actions) or the environment may be

stochastic (*i.e.* an environment with random events).

- **Imperfect information.** The environment is only *partially observable* (*i.e.* part of the environment is hidden to us). Moreover, other agents may be able to observe the information hidden to the modeller.
- **Dynamic behaviour.** Other agents may change their behaviour over time. Examples of this include customers changing their purchasing habits over time or a game player changing their strategy over the course of one or more matches.

Depending on the domain, a modelling agent may need to account for any of these features. Stochastic observations and dynamic behaviour require the agent to plan based on the expected outcome of future decisions and environmental changes rather than a deterministic outcome. Imperfect information makes it difficult to associate an agent's observed behaviour with the state they were in when they made the decision. The combination of stochastic observations, imperfect information, and dynamic behaviour amplifies the challenge posed by each of these features. When combined, these features make it difficult to determine if an agent's observed behaviour is due to their stochastic decisions, a change in their behaviour, or some element of the hidden environment. Finally, the variability in possible future outcomes caused by these features make it difficult to form a reliable agent model with only limited observations. Our work presents an agent modelling technique which addresses all of these challenges.

1.2 Thesis Contributions

In this thesis we explore the use of *state estimation* techniques to address the four challenges of agent modelling: limited observations, stochastic observations, imperfect information, and dynamic behavior. State estimation tracks a stochastic process' hidden state variables by observing noisy functions of these variables. Such techniques represent uncertainty in the hidden state as a probability distribution and use a recursive Bayes' formula to update the belief with each new observation. Although these techniques require a complete probabilistic model of the process dynamics and observations, techniques for *dual estimation* have relaxed this requirement by inferring these models from data at the same time as estimating an agent's state. To validate our use of state estimation techniques for agent modelling, we demonstrate the algorithms in the domain of Kuhn poker (a simplified variant of poker that will be explained further in Section 2.5.1).

The primary contribution of this work is our application of state estimation techniques for agent modelling. State estimation has received little attention in the agent modelling literature, yet it appears well-suited to the agent modelling task. This work demonstrates that state estimation is, in fact, well-suited for agent modelling. Our experiments show that in addition to effectively modelling static agents, state estimation lends itself nicely to modelling dynamic agents when the agent's dynamics are known.

The second major contribution of this work is our use of dual estimation for inferring an agent's dynamics from data. We show that given the correct form of an agent's dynamics model, we can learn the model's unknown parameters. Furthermore, our experiments show that dual estimation makes our approach robust to unknown model parameters and even an incorrect form of the dynamics model.

Our experiments show the efficacy of state estimation for agent modelling. In addition to improving upon current techniques for agent modelling in Kuhn poker, the state estimation framework is general enough to be applicable in many other domains. Moreover, the framework provides a direct approach for modelling dynamic agents – a challenge which is often at most an afterthought in many agent modelling techniques.

1.3 Thesis Outline

This thesis is presented as follows. Chapter 2 discusses work related to agent modelling. Chapter 3 presents background material on game theory and the theoretical foundations of state estimation including the state estimation algorithms used in the thesis. Chapter 4 presents our application of state estimation algorithms to Kuhn poker including further description of how dynamic opponents are handled. In Chapter 5, we describe experiments against a variety of dynamic computer agents in different scenarios. Finally, Chapter 6 concludes and provides directions for future research.

Chapter 2

Related Work

Agent modelling has been explored under a number of different names and in a variety of domains. In the artificial intelligence literature, problems like policy recognition, behaviour recognition, and opponent modelling are all connected to the agent modelling problem. Before we explain our application of state estimation to agent modelling, we will review some of the past work in agent modelling. Each of the techniques presented represents a method for modelling agents in certain domains. Although our approach bears similarities to many of these techniques, none of the previous approaches handle the four agent modelling challenges to the same extent as our state estimation approach. In particular, our approach to handle an agent's dynamic behaviour is distinct from the approaches mentioned here. We begin by describing related work in agent modelling and we will conclude with a comparison of the different approaches to agent dynamics (Section 2.6).

2.1 Model Recognition

Agent modelling research in the areas of behaviour recognition, policy recognition, and plan recognition all share a common feature. In general, each of them attempts to recognize an agent's observed behaviour as one of a discrete set of agent behaviours. This discrete set, or *library*, of agent behaviours is typically specified by hand before recognition takes place and is usually relatively small. We will refer to algorithms that fall under these related domains as *model recognition* algorithms.

To contrast our work with this research we present two model recognition algorithms. The algorithms we present are similar to our approach in that they handle stochastic observations and imperfect information (though this is not true of all model recognition algorithms). In contrast to model recognition in general, our work uses a parameterized continuum of agent behaviour rather than a library of discrete behaviours.

Behaviour Hidden Markov Models

One example of a model recognition algorithm is presented in the behaviour recognition work by Han and Veloso [11]. Their work attempts to recognize the behaviours of other agents using **hidden**

Markov models in the domain of robot soccer. Before going into the details of their approach, we briefly define the structure of a hidden Markov model.

Formally, a *hidden Markov model* (HMM) can be represented by a 5-tuple $\lambda = \{S, Z, A, B, \pi\}$ where $S = \{s_i\}$ is the set of hidden states, and Z is the space of possible observations. Letting X_t represent the unobservable state of the HMM at time t, A is a transition probability matrix where $a_{ij} = \Pr(X_{t+1} = s_j | X_t = s_i)$, $B = \{b_i(z) = \Pr(z | X_t = s_i)\}$ is the set of observation probabilities (where $z \in Z$), and π is the initial state distribution where $\pi_i = \Pr(X_1 = s_i)$.

Han and Veloso represent each behaviour in their library with a separate Behaviour HMM (BHMM). A behaviour HMM is effectively a HMM where the states correspond to different stages of recognizing the specified behaviour. The states are divided into four categories: initial states, accept states, intermediate states, and reject states. Consider the behaviour of getting a graduate degree. Initial states represent the set of states which an agent would start at for a given behaviour (*e.g.* applying for school). Accept states indicate when an agent has completed the given behaviours (*e.g.* thesis submitted). Intermediate states represent sub-behaviours that need to be performed to complete the high level behaviour (*e.g.* taking classes, doing research, writing a thesis). Finally, reject states indicate behaviour that is contrary or unrelated to performing the given behaviour (*e.g.* dropping out of school, travelling for three months).

To recognize what behaviour an agent is performing, a separate BHMM is instantiated for each behaviour to test for. Moreover, because an agent may not start executing a behaviour at the same time the BHMMs are instantiated, new BHMMs are instantiated at regular intervals to ensure one of them is started close to when the agent begins. After instantiation, the probabilities for the states of a BHMM are computed using the *forward algorithm*. Specifically, compute the following probability.

$$\Pr(X_t = s_i | z_1, \ldots, z_t, \lambda) = \frac{\alpha_i(t)}{\sum_j^{|S|} \alpha_j(t)} \qquad \text{where} \quad \alpha_j(t) = \sum_i^{|S|} \alpha_i(t-1) a_{ij} b_j(z_t)$$

To prevent the number of BHMMs from growing without bound, a BHMM is removed when the most likely state is a reject state with a probability above a specified threshold or if a specified maximum time for the behaviour to run is exceeded. With the BHMMs running concurrently, we can examine the probability of each BHMM's accept states to determine the probability that the agent is performing the specified behaviour.

Abstract Hidden Markov Models

Previous efforts in policy recognition by Bui and colleagues [7] also attempt to recognize an agent's behaviour. Unlike Han and Veloso's work, which use a BHMM for each behaviour in their library, this work uses a hierarchy of behaviours to represent their entire library. This work attempts to recognize another agent's behaviour using a Rao-Blackwellized particle filter (which we discuss in Section 3.2.3) combined with a dynamic Bayesian network to simultaneously compute the probability of each behaviour at each level in the hierarchy.

More specifically, it is assumed that the agent being modelled behaves according to a hierarchy of behaviours[1]. Consider the example presented in Section 3.3 of [7] for recognizing the behaviour of people in a building. Behaviours at the bottom of the hierarchy consist of single step actions (*e.g.*move left). Going up the hierarchy results in increasingly abstract behaviours. For instance, the second, third, and fourth level policies could indicate which door to exit from in the room, wing, or entire building, respectively. Although this hierarchy provides a convenient way to compose behaviours for the library, the resulting set of possible behaviours is still discrete and must be provided before recognition can take place.

The specific formulation of the hierarchy's abstract policies ensures that a behaviour only terminates if all of the behaviours below it in the hierarchy have also finished. The authors use this fact to construct a dynamic Bayesian network (DBN). Each level of the hierarchy (aside from the top and bottom level) has two variables in the DBN: one to represent the policy used at that level, and one to indicate if the policy finished on the current time step. The DBN also includes variables for the state and observations (though observations can be omitted if the state is not hidden).

Performing inference in a DBN of this size is typically intractable. Due to the authors' assumption that a given policy can only terminate when all policies below it have terminated, they are able to gain some conditional independence properties. Specifically, they observe that variables in the current time slice are conditionally independent given the entire state history and the start and end times of each of the policies. Given this information, updating the beliefs for each level of the policy hierarchy becomes feasible.

Unfortunately it is impractical to have this information in general. Therefore, the authors employ a Rao-Blackwellized particle filter (which we discuss in Section 3.2.3) to draw weighted samples of the variables necessary to make the DBN computation tractable. The probabilities of the policies are then computed from the tractable DBN structure using the samples. The weighted average of each sample's resulting policy probabilities is the resulting estimate for each policy's probability given the sequence of observations.

2.2 Model Learning

Another way to model agents is to learn a model of their behaviour from observations rather than trying to recognize their behaviour from a library. We call this type of algorithm a *model learning* algorithm. Unlike model recognition algorithms, model learning algorithms generally make some assumptions about the general form of an agents' behaviour and then they try to learn the agent's parameters given the assumed form.

There are many algorithms for learning an agent model and they each have their own strengths and weaknesses. Section 2.5 describes a number of model learning techniques that were applied to

[1]Formally, they assume agents make decisions according to an abstract Markov policy (AMP) model. (also known as *options* or *policies of Abstract Markov Decision Processes*).

variants of poker. Like our approach, these techniques are able to model imperfect information and stochastic decisions. On the other hand, some model learning techniques do not address either of these challenges. We present one such technique by Carmel and Markovitch.

Modelling Agents as Deterministic Finite Automata

Carmel and Markovitch's work proposed an algorithm for modelling an "opponent" in a game as a **deterministic finite automata** (DFA) [8]. A DFA is a finite state machine where each state-input pair has exactly one transition to the next state.

Using observations of the opponent, their algorithm iteratively builds a table, T, representing the opponent's DFA. Let t be the current time step. Furthermore, let Σ denote the set of possible observations the opponent can make on any time step (including the null observation λ). The authors assume that they receive perfect information about the world (*i.e.* the world can be fully observed including the opponent's actions). Therefore, each observation of the opponent consists of the opponent's full observation history up to the current time step $h_{1:t}$, and the opponent's action or output, o_t. The rows of T represent the elements of $R = \{h_{1:j}\sigma | \sigma \in \Sigma, 1 \leq j \leq t\}$ which correspond to all observed histories and their single observation extensions. The columns correspond to elements of Σ (although not all possible observations necessarily have a column). Upon receiving the observation for time t, the table is expanded by letting $R = R \cup \{h_{1:t}\sigma | \sigma \in \Sigma\}$ and setting the table entry $T(h_{1:t}, \lambda) = o_t$. If the current observation is inconsistent with the DFA represented by T (*i.e.* there are two elements in R, say r_1 and r_2, that are represented by the same state in the DFA but $T(r_1, \sigma) \neq T(r_2, \sigma)$ for some $\sigma \in \Sigma$) then their algorithm attempts to resolve the inconsistency. Resolving inconsistencies is done in two ways. The easiest way is to add new states to the DFA. The second way is to change the entries in the table which are not directly supported by an observation of the opponent. Although this does not necessarily represent the opponent's true DFA, these entries have no fixed value because they have yet to be observed. This approach has the benefit that it keeps the DFA compact until an observation causes an inconsistency.

Because DFAs have no mechanism for handling non-determinism, the underlying assumption that an opponent can be modelled as a DFA results in this technique being incapable of modelling stochastic observations or imperfect information. Furthermore, it is not obvious how this technique could be extended to handle these challenges or how to scale up the algorithm to large state spaces.

2.3 Imitation Learning

As with model learning algorithms, imitation learning attempts to build a model of another agent's behaviour from observations of the agent. Unlike a general model learning agent, an imitation learning agent (or *imitator*) assumes that it can learn how to behave from other agents in the environment. If other agents have similar objectives, then an imitator can determine how to act by observing the other agents and following a similar behaviour. There are several techniques that can be used to perform imitation learning. For example, an agent's reward function can be learned, yielding information about the states an agent prefers. This is the approach taken by inverse reinforcement learning (which is covered in Section 2.4). Other techniques use observations of other agents to inform themselves about the state dynamics of the world. Price and Boutilier take this approach in [21]. This work uses Bayesian statistics in a reinforcement learning MDP framework to improve an agent's knowledge of the MDP's dynamics.

Specifically, the authors assume that the probability distribution over the MDP's dynamics can be factored into the product of Dirichlet distributions[2]. Each state s in the MDP has a corresponding Dirichlet distribution over the possible next states s'. Because Dirichlet distributions are conjugate priors, the authors are able combine three Dirichlet distributions to derive the state dynamics. These three distributions represent: the imitator's prior belief about the dynamics, the imitator's own experiences of the MDP's dynamics, and the imitator's observations of other agents' experiences. An imitator using this technique chooses actions according to their value function over state-action pairs. Then, upon observing their own state transition and the transitions of other agents, the imitator updates the Dirichlet distribution with the observed outcomes. Finally, the imitator updates its value function according to the new beliefs about the dynamics and the imitator chooses its next action.

Although this work does solve some of the four agent modelling challenges, it does not present a formulation for imitation learning when the other agents are only partially observable. Price and Boutilier do mention that it can be extended to solve this problem but they state it is considerably more complex. Furthermore, it fails to address the case when the behaviour of other agents is dynamic (which will be discussed further in Section 2.6).

2.4 Inverse Reinforcement Learning

The inverse reinforcement learning (IRL) problem aims to extract an agent's reward function R from the agent's policy π or samples thereof. Knowing an agent's reward function would reveal what outcomes they prefer. Moreover, if we can extract an agent's reward function, then it can be used to generate a new agent that would imitate the agent's behaviour. Ng and Russell present linear programming techniques for solving the IRL problem [19].

In the case where the MDP is known except for the reward function (*i.e.* we know the sets of states and actions, the state transition probabilities, and the discount factor), the authors derive the linear constraints necessary for R to be the reward function for the MDP's optimal policy. They also provide linear penalty terms that elicit reward functions which are both "meaningful" (*i.e.* $R \neq 0$) and "simple" (*i.e.* R is non-zero in only a few states).

The authors also extend their linear programming technique to solve the IRL problem in large or infinite state spaces. They assume that the reward function can be approximated by a linear combination of fixed basis functions (*i.e.* $R = \alpha_1\phi_1(s) + \cdots + \alpha_d\phi_d(s)$) and that they know these

[2]A Dirichlet distribution is the multivariate generalization of the Beta distribution presented in Section 4.3.1

basis functions. Using this information, the authors can create a linear program to fit the reward function's coefficients. Unlike their first approach, the large (and potentially infinite) state space prevents all possible constraints of the MDP from being included in this solution (since there would be constraints for each state of the MDP). Therefore, the solution only satisfies the constraints on a set of samples from the state space rather than the full state space. Finally, the reward function for the MDP's optimal policy may not be expressible as a linear function. To solve this problem, additional penalty terms are added to the linear program that relax the optimality constraint while ensuring that R remains near-optimal.

A more realistic case where we would want to solve the IRL problem is when we only know about a policy from sample trajectories through the MDP. For this case, the authors iteratively refine R by sampling trajectories through the MDP. As before, it is assumed that R can be expressed as a linear function of known fixed basis functions. Specifically, on iteration k of the algorithm, sample trajectories are computed for π^* (the assumed optimal policy) and each policy in $\{\pi_1, \ldots, \pi_k\}$ (on the first iteration π_1 is randomly chosen). The samples for each policy are used to compute $\hat{V}_i^\pi(s_0)$: the average empirical return assuming that $R = \phi_i$. Using these values, the authors estimate each policy's expected return $V^\pi(s_0)$ as $\hat{V}^\pi(s_0) = \alpha_1 \hat{V}_1^\pi(s_0) + \cdots + \alpha_d \hat{V}_d^\pi(s_0)$. The authors then formulate a linear program that maximizes the difference between the estimates for π^* and π_i by adjusting the reward function coefficients $\alpha_1, \ldots, \alpha_d$. Once this is done, a new policy π_{k+1} is generated that maximizes $V^\pi(s_0)$ under the new R. π_{k+1} is then added to the current set of policies $\{\pi_1, \ldots, \pi_k\}$ and the process repeats until we find an R that we are "satisfied" with.

In general, these techniques assume that many aspects of the agent being modelled are known. For general agents, it is unreasonable to expect that we would know an agent's transition probabilities. Moreover, it is not obvious that sample trajectories of an agent's policy could be generated or that we would have correct basis functions for an agent's reward function. Even if these basis functions were available, if an agent's state was only partially observable, the modelling agent would not be able to compute the agent's expected return as easily. Marginalizing over the unknown parameters could resolve the partial observability problem, but we see no obvious solution to having an incorrect set of basis functions. We now turn our attention to our experimental domain of poker and the agent modelling techniques that have been used in that domain.

2.5 Poker Research

In recent years the interest in poker and poker research has increased dramatically. From a scientific standpoint, poker provides an interesting and challenging testbed for AI research [4] that embodies all four agent modelling challenges. The environment and often the players are stochastic. Each player has private information which may never be revealed. Finally, dynamic behavior is an integral part of human players and expert poker tactics. For example, human players will adapt their behaviour over time as they model their opponents and learn their own weaknesses. Also, mislead-ing other players by projecting a false style of play can create a "table image" that can exploited by "switching gears" to a strategy contrary to the "table image". This is a common tactic and involves drastic style changes.

Agent modelling is expected to play a key role in the eventual development of world champion poker-playing programs. Currently, many of the competitive poker playing programs employ a game-theoretic pseudo-equilibrium approach without any agent modelling [3, 10]. By failing to identify and exploit their opponents' weaknesses, these approaches are unlikely to best the world's top human players. This was also the observation of a world class player after evaluating one of these game theoretic programs, "You have a very strong program. Once you add opponent modelling to it, it will kill everyone" [5]. Although equilibrium techniques can be challenging for humans, their static nature allows human players to probe the program for weaknesses without the threat of an adapting opponent. Poker's inherent challenges and the expected need for agent modelling techniques to create a world champion poker-playing program make this domain ideal for examining the problem of dynamic agent modelling.

The University of Alberta Computer Poker Research Group (CPRG) has done extensive research on techniques for building computer poker agents. Most of this research has focussed on Texas Hold'em poker – a variant of poker that is much more complex than Kuhn poker. We will first discuss some of the previous approaches to agent modelling in Texas Hold'em poker and then discuss our experimental domain of Kuhn poker.

Loki & Poki

The CPRG's early agent modelling techniques use a frequentist model of past behaviour to infer an agent's future actions and possible hidden cards. Their first opponent modelling efforts were seen in *Loki*, a program for multiplayer Texas Hold'em. Loki learned an agent model by updating separate frequency tables of 36 different game contexts based on each opponent's observed play [6]. There was a context for each combination of actions (fold, call, raise), action cost (zero, one, or more than one), and betting round (pre-flop, flop, turn, river). This information was used to infer the mean and variance of an opponent's expected hand strength[3] in the specific game context. The mean and variance values were used to reweight the distribution over the opponent's hidden cards. This enabled the modeller to more accurately infer its odds of winning the hand.

Loki was later refined and renamed *Poki*. The refinements to Loki's opponent modelling system were based on the use of artificial neural networks (ANNs) to predict an opponent's future actions [9]. In this work, an ANN was trained offline from logs of human play. The ANN used properties of the game context for inputs and a node for fold, call, and raise in the output. Unfortunately, this technique was too slow for online learning. Instead of using the ANN online, the weights of the ANN input features were used to determine which game contexts were important factors for

[3]Expected hand strength is a metric of the quality of a player's cards.

determining an opponents actions. This yielded two new contexts that were incorporated into Loki's frequency tables: previous action and previous amount to call.

Vexbot & BRPlayer

Subsequent opponent modelling efforts in heads-up (*i.e.* two player) Texas Hold'em also used frequentist opponent models, except with considerably more contexts. The CPRG's foremost opponent modelling program, Vexbot, and its successor BRPlayer, both use a frequentist opponent model to drive an imperfect information game tree search.

Vexbot [5] and BRPlayer [22] search the imperfect information game tree using the miximix algorithm. Miximix is an extension of the expectimax algorithm for stochastic game tree search to imperfect information domains. Miximix computes the expected value (EV) of decision nodes in the game tree by treating an opponent's unknown information as a chance node. To compute a decision node's EV, miximix must know two things: the probability of the opponent's actions at each of its decision nodes, and the EV of the game tree's terminal nodes.

Of course, this kind of detailed information about our opponent is unknown. Instead, Vexbot and BRPlayer estimate these values by learning similar opponent models. The opponent model is learned by tracking two distinct statistics for estimating these values. First, to estimate the opponent's action probabilities, the model stores the frequency of an opponent's actions at each of the possible betting histories (*i.e.* at each of the decision nodes in the game tree). Note that in these models the betting histories are not conditioned on any chance events (including publicly known cards).

Next, we need to estimate the EV of a terminal node. Two separate cases are considered for this computation. If the terminal node is due to a player folding their hand, then the EV of that node can be computed exactly from the betting history that led to the node. Otherwise, the players have a *showdown* and the the player with the strongest hand wins the money that was bet (*i.e.* the *pot*). To compute the EV of a showdown, the opponent model needs to estimate the probability of winning the hand. Therefore, the modelling agent must be able to estimate the probability that the opponent's hidden cards are stronger than its own. One might consider using a list to store a measure of the opponent's hand quality (*e.g.* expected hand strength) for each time the opponent was observed at each showdown node. Rather that storing these lists, the opponent's hand quality is abstracted into discrete groups called *buckets*. A histogram of observed opponent buckets is stored at each showdown node.

To accommodate an opponent's dynamic behaviour, Vexbot gives preference to more recent observations by gradually "forgetting" old observations. This is done by weighting the history of observations according to an exponential decay function. Although this decay function was not included in BRPlayer, it could be easily added. Letting h be the history decay factor, Vexbot gives the most recent observation a weight of $(1 - h)$, the previous $1/(1 - h)$ observations a weight of $(1 - 1/e)$, and so on.

On each hand, the opponent model's frequency counts are updated based on the observed outcome. When the miximix search is computing a decision for the modelling agent, the probabilities needed by miximix are derived from the relative frequency of the observed outcomes. The main difference between Vexbot and BRPlayer lies in this derivation. In order to learn faster, both Vexbot and BRPlayer combine showdown observations from different betting histories by weighting the related observations according to several similarity metrics. BRPlayer considers more of these similarity metrics than Vexbot does. After the expected value of a decision node is computed, the resulting EV is used to determine the next action. In the case where miximax is used (a special case of miximix that chooses the action that maximizes EV) this will result in a deterministic action. Miximix, on the other hand, yields less predictable strategies by returning a probability distribution over possible actions weighted by their EVs.

Although these techniques are able to handle stochastic observations, imperfect information, and dynamic behaviour, they require a considerable amount of data to be effective. In experiments with BRPlayer modelling the CPRG's static pseudo-equilibrium player, PsOpti4, BRPlayer took between 20,000 to 175,000 hands before it was able to break even. It is unreasonable to expect that a human player would ever play this long.

Another vulnerability with these techniques is that the opponent model can be easily mislead. Because the showdown EV calculations are based solely on what the modelling agent has observed, it can fail to account for the rules of the game. Consider an opponent who gets a lucky streak of cards and they repeatedly play to a showdown. The opponent model would grow to believe that the opponent both never folds and always has good cards at a showdown. This situation could result in a "folding trap" where the model believes it is always best to fold since my opponent only goes to the showdown and always has good cards. Of course, the opponent cannot always have good cards due to the random nature of the deck. Unfortunately, Vexbot and BRPlayer are not aware of this.

Bayes' Bluff

The CPRG's most recent approach to modelling opponents in heads-up Texas Hold'em combines Bayesian techniques and Monte Carlo sampling to infer an opponent's current strategy [23].

This work, named *Bayes' Bluff*, made three contributions to modelling agents with Bayesian inference in games similar to Texas Hold'em. First, the authors demonstrated how to compute the probability of an individual observation given the opponent's strategy regardless of whether the opponent's hidden cards are revealed at a showdown. This derivation was necessary for their second contribution which showed how to compute the posterior distribution over opponent strategies given a set of past observations. Finally, the authors show that an algorithm similar to expectimax will compute the Bayesian best response to the posterior distribution over opponent strategies.

Unfortunately, computing the exact Bayesian best response to the posterior requires the evaluation of an integral over opponent strategies. This is prohibitively expensive except in small games with relatively few observations. To circumvent this problem, the authors present three sampling techniques to approximate the exact Bayesian best response. First, the integral itself could be approximated by sampling the prior over opponent strategies. Second, the maximum a posteriori (MAP) response could be computed instead of the Bayesian best response. This is also computationally expensive, but it can also be approximated by sampling the prior and using the strategy that is most probable as the MAP strategy of the opponent. Finally, Thompson's response can be used. This technique samples the prior distribution and uses importance sampling (covered in Section 3.2.2) to compute each sample's posterior probability. A strategy is then chosen from the set of samples based on their posterior probability.

The techniques that we present in this thesis are very related to some of the techniques used in Bayes' Bluff. Both approaches use Bayesian inference, with Monte Carlo sampling. In fact, our technique is analogous to Bayes' Bluff when using samples to estimate the integral over opponent strategies. The key difference between the two approaches is that Bayes' Bluff assumes an opponent is static, never changing their strategy throughout the game. Our work, on the other hand, explicitly models dynamic agents.

2.5.1 Kuhn Poker

We will now focus on our experimental domain of Kuhn poker. Kuhn poker is a toy variant of poker developed by Dr. Harold Kuhn. It is a zero-sum game involving two players; two actions, *bet* and *pass*; and a three card deck, containing a jack (J), queen (Q) and king (K). Each player is dealt one card privately and the third card is set aside unseen. The first player may then either bet or pass. If the first player bets the second player may either bet, causing a *showdown*, or pass, to *fold*. If the first player passes, the second player can also pass, causing a showdown, or bet, forcing the first player to make a final decision of either bet, for a showdown, or pass, for a fold. In the case of a fold, the non-folding player wins one dollar from the folding player. In the case of a showdown, the player with the higher card (king is high, jack is low) wins one dollar if neither bet, or two dollars if both players bet.

Kuhn poker's small size and diverse strategies make it an ideal domain for our investigation. It preserves many strategic properties found in larger variations of poker played by humans, yet a game theoretic analysis of the game is tractable and exists [16]. For instance, two common poker tactics are *trapping* (*i.e.* acting as though your hand is weaker than it truly is) and *bluffing* (*i.e.* acting as though your hand is stronger than it truly is). Kuhn poker preserves both of these properties. Betting with a jack represents a bluff while passing with a king is a trap. In addition to preserving strategic properties, Kuhn poker also has a known parameterization for players' strategies, and computing the best-response to a given strategy is straightforward. Since these features are all active research directions in full versions of poker, Kuhn poker offers a clean domain for evaluating ideas. Moreover, as we discuss next, it has already been a testbed for agent modelling research.

Hoehn and colleagues [12, 13] have recently worked on agent modelling techniques for short-term modelling of static opponents in Kuhn poker[4]. Hoehn examined a variety of different agent modelling techniques including explicit modelling using frequentist statistics and implicit modelling using experts algorithms. We present his approaches here and compare the performance of our technique with Hoehn's in Section 5.1.

Explicit Modelling

Like the aforementioned model learning algorithms, Hoehn's explicit modelling technique attempts to learn the parameters of an agent model from observations. In this case, the parameters represent an opponent's strategy in Kuhn poker.

In many regards, Hoehn's explicit modelling is similar to the approach taken with Vexbot and BRPlayer. Both techniques use frequency counts to estimate an opponent's action probabilities. Unlike Vexbot and BRPlayer, which store frequency counts for observed betting strings and abstracted showdown outcomes, Kuhn's small size allows Hoehn to store frequency counts for all possible observations of the betting history and chance events (from the modelling player's perspective).

Hoehn also derives closed form equations for estimating the frequency counts of observations where the opponent's cards are unknown. These estimations take into account the observation frequencies, the probability of the random cards, and the modeller's own strategy. This gives Hoehn's explicit modelling one distinct advantage over Vexbot and BRPlayer: it knows about the rules of the game. From the combined information provided by the frequency counts and the frequency count estimations, Hoehn is able to estimate all of the parameters of an opponent's strategy.

To build a functional computer agent using this agent model, Hoehn uses his parameter estimation technique together with a variety of exploration strategies and a best response computation identical to the one we present in Section 4.4.

Although this technique does allow for the rules of chance events to be integrated into the agent model, the technique would be very difficult to scale to a game like Texas Hold'em poker.

Implicit Modelling

In contrast to explicit modelling, Hoehn's implicit modelling does not attempt to directly learn the parameters of an opponent's strategy and then compute a counter-strategy. Instead, implicit modelling attempts to identify a good counter-strategy to the opponent while being unaware of the opponent's specific weaknesses or vulnerabilities.

This is done through the use of the regret minimization techniques presented by Auer and colleagues [1]. Regret minimization algorithms present a technique for combining the advice of a set of *experts*. The foundation of regret minimization lies in two algorithms: *Hedge* and *Exp3*. We will discuss each of these algorithms next.

[4]Technically Hoehn's work is done in the domain of undominated Kuhn poker, which we discuss further in Section 4.1

The Hedge algorithm requires a set of experts and a reward associated with each expert. At time step t of the Hedge algorithm, expert i with cumulative reward $G_i(t-1)$ over the last $t-1$ time steps is chosen from the set of K experts with probability $p_i(t) = \exp(\eta \mathrm{G_i(t-1)}) / \sum_{\mathrm{j=1}}^{\mathrm{K}} \exp(\eta \mathrm{G_j(t-1)})$. η is a parameter to Hedge that controls the learning rate (*i.e.* the emphasis given to using an expert with largest cumulative reward). The chosen expert would then be used to make decisions on time step t. Hedge was designed as a regret minimization technique for games where it was possible to determine the reward that each expert would have received if they were chosen at a given time step. It is also central to other experts algorithms, such as Exp3, which are designed to handle the case when it is not known what reward the other experts would have received.

Exp3 handles the case when an expert's reward is unknown by generating simulated rewards for the experts. Simulating rewards effectively reduces this case to the full information situation that Hedge was designed for. The Exp3 algorithm, which stands for Exponential-weight algorithm for Exploration and Exploitation, does as its name suggests. First it invokes Hedge to get the exponentially weighted probabilities $p_i(t)$ of choosing expert i for exploitation on the current time step t. Then, it modifies the probabilities by adding in a specified amount of uniform exploration, γ. Expert i is chosen according to the resulting probability $\hat{p}_i(t) = (1-\gamma)p_i(t) + \gamma/K$. This expert is then used to make decisions and a reward $r_i(t)$ is received. To give Hedge full information, Exp3 generates simulated rewards of $\hat{r}_j(t) = 0$ for $j \neq i$, and $\hat{r}_i(t) = r_i(t)/\hat{p}_i(t)$ for the chosen expert. Scaling the observed reward in this manner compensates experts that are chosen infrequently. Formally, it ensures that the expected value of an expert's simulated cumulative reward after T time steps is equal to their cumulative reward with full information (*i.e.* $\mathrm{E}[\sum_{t=1}^{T} \hat{r}_j(t)] = \sum_{t=1}^{T} r_j(t)$). Exp3 is an easily implemented algorithm that has useful theoretical guarantees for performance. Moreover, Exp3's basic framework provides the foundation for more sophisticated experts algorithms, such as Exp4, that generate simulated rewards which are more representative of the true rewards.

Hoehn's work presents and examines the effectiveness of several such improved experts algorithms along with Exp3 [12]. In general, Exp3 can be modified to learn more quickly by sharing rewards between experts. When expert j has a non-zero probability of acting in the same way as the chosen expert i, expert j had some probability of obtaining the same reward as expert i. In this case, the reward received by expert i is scaled by the probability that any of the experts would have made the same decisions as i. Each of the agreeing experts is given a share of the rewards proportional to their contribution to the overall probability of making the decisions taken by the chosen expert. This style of reward sharing has been investigated by Auer and colleagues [1] as the Exp4 algorithm. Hoehn's SharingExp3 algorithm also uses this approach and has been shown to yield better results for modelling agents in Kuhn poker.

Using regret minimization techniques for implicit modelling has both advantages and disadvantages. Because the algorithms only consider the reward received by the chosen expert, the opponent's hidden cards do not need to be directly accounted for by the algorithm. Furthermore, aside from using the outcomes of chance events to determine the probability of another expert making the same decisions as our chosen expert, the implicit modelling techniques do not need to know about the stochastic observations. This makes regret minimization algorithms an enticing approach to agent modelling that provides theoretical guarantees on performance. Unfortunately these implicit modelling techniques also have the drawback of being relatively slow to learn. Sharing rewards between experts mitigates this to some degree, but as Hoehn's experiments indicate, none of his implicit modelling techniques fared as well as his explicit modelling algorithm.

Despite some weaknesses, Exp3 is a relatively simple algorithm for modelling dynamic opponents. Because of this, we revisit Exp3 in Section 5.3 where we use this algorithm as an example of a non-oblivious opponent. We will now conclude this chapter with a discussion on the different agent modelling techniques for modelling dynamic agents.

2.6 Modelling Agent Behaviour Dynamics

One significant difference between our work and previous work is how we handle an agent's dynamics. Although some of the techniques just described are capable of modelling dynamic agents, the assumptions that are made about the dynamics can be very restrictive. To clarify the differences in how dynamic agent modelling is addressed by other techniques and our own, we will discuss the dynamic agent modelling capabilities of each of the aforementioned approaches. We will then introduce the strengths and limitations of our own technique before moving on to its description.

Much of the previously described work on agent modelling makes the assumption that an agent's behaviour is static in some way. The behaviour HMMs presented by Han and Veloso [11] assume that the agent does not switch between behaviours part way through recognition. Abstract HMMs [7] only consider one top-level policy at a time and are unable to model interleaved plans. Price and Boutilier's imitation learning algorithm [21] explicitly assumes that the behaviour of the agent being modelled is static. The same is true for Ng and Russell's inverse reinforcement learning work [19].

All of these techniques can model behaviours that are, in a sense, dynamic. Specifically, they can recognize behaviours that involve a sequence of distinct actions. Consider the example presented by Bui and colleagues of a person navigating through a building to an exit. This agent is "dynamic" because the behaviour of "exit through the east door" may consist of different sub-behaviours like "enter the hallway", "go down the stairs", and "enter the east lobby". This is not the type of behaviour dynamics that we are interested in. We are interested in behaviour dynamics that allow for an agent to switch between multiple behaviours that can take entirely different actions despite being in the same situation. Returning to our example, consider that the person typically leaves through the east exit but occasionally leaves with a friend from the west exit. Moreover, they are a little forgetful and often walk to the east lobby before realizing they need to head to the west door. It is in these situations, where an agent may change their high level behaviour according to some unknown process, that we want to be able to model an agent and their dynamics effectively.

Some of the previous work handles this type of dynamics but with some specific assumptions about how the unknown dynamics process works. Carmel and Markovitch's opponent modelling work [8] can represent dynamics agents, but only if their behaviours dynamics can also be modelled as a DFA. Realistically, this type of dynamic behaviour is not any different from the previous "dynamics" because the larger DFA could be considered as a single, albeit more complex, behaviour.

Frequency count approaches to opponent modelling in poker (Loki, Poki, Vexbot, BRPlayer, and Hoehn's explicit modelling) provide a straightforward technique for modelling static agents. Although these algorithms could be extended to address dynamic opponents by decaying the frequency counts over time, as was done with Vexbot, this amounts to "forgetting" an agent's past behaviour in the hope that they will behave similarly to recent actions. If an agent's future behaviour is highly dependent on their past behaviour, this type of information decay will discard valuable information. In our example of a person leaving a building, we might be able to infer that if the person is heading for the east exit and they stop, then they are more likely to be heading to the west exit. But, if we are decaying information, this pattern may never be revealed.

Regret minimization algorithms, such as Hoehn's implicit modelling algorithms [12], can be used to model dynamic agents. Unfortunately, they only adapt to a change in an agent's behaviour through the rewards they receive. Without specific mechanisms to learn the parameters of an agent's dynamics, they are only able to adapt in reaction to an agent's behaviour rather than active prediction of an agent's next behaviour.

Our work provides mechanisms to explicitly model an agent's dynamic behaviour. This is, in many ways, an extension of the ideas from Bayes' Bluff to handle dynamic opponents. Along with the parameters of an agent's behaviour (which is a strategy in Kuhn poker) we store additional parameters for describing an agent's dynamics. On their own, these parameters can be used to describe how an agent's behaviour changes. We also provide a technique for dual estimation that allows us to estimate these parameters as we observe our opponent's behaviour.

Although our technique provides a method for modelling dynamic behaviour, there are still limitations on the type of behaviour dynamics that we can model. As is described in Chapter 4, our technique uses two parameters that describe an agent's behaviour dynamics. Our choice of state variables and dynamics models restricts our agent to modelling changes in behaviour that are only dependent on the previous behaviour (*i.e.* we assume the dynamics are Markovian). Our technique does not require Markovian dynamics in general, but the computational cost of modelling non-Markovian dynamics grows rapidly with the number of previous time steps modelled by our algorithm[5]. Despite this limitation, we will show that our state estimation techniques provide effective and robust modelling of dynamic agents in the domain of Kuhn poker.

[5]The agent's state in each previous time step would need to be tracked to model non-Markovian dynamics. The number of particles required by our state estimation algorithms to cover (and therefore accurately estimate) the state space grows exponentially with the size of the state space.

Chapter 3

Background

Describing our application of state estimation to Kuhn poker requires some preliminary knowledge in game theory and state estimation algorithms. We begin by presenting the essential game theory knowledge used by our work in Section 3.1. We conclude our background content with an introduction to state estimation algorithms in Section 3.2.

3.1 Game Theory

Game theory is a branch of mathematics that studies strategic interactions between agents. The theoretical basis of game theory provides a useful framework for our exposition. Although we present some key ideas of game theory, this discussion only presents the concepts needed for our application of state estimation to Kuhn poker. The formalism for this section is inspired by Owen [20]. For a more complete treatment of game theory, see this or any other game theory text.

In general, a **game** refers to a process where two or more agents, called **players**, are given rewards based on making alternating decisions using whatever information is available to them (which may be imperfect). To accommodate for random events (*e.g.* random cards), games may also include an agent that makes stochastic moves.

More precisely, players are asked to make decisions at information sets corresponding to the information observed by the player. An **information set** I_p for player p corresponds to the set of all possible decision nodes where player p has the same public and private information. In a perfect information game, each information set is distinct. For imperfect information games, multiple decision nodes can correspond to a single information set. For example, in Kuhn poker, each possible information set for player one's first decision contains two decision nodes: one for each possible unknown card that the opponent may hold.

Using this concept of information sets, we can now define a player's strategy. A player's **strategy** defines how they play the game. A strategy $\sigma_p \in \Sigma_p$, where Σ_p is the set of all strategies available to player p, is a function that maps information sets to a distribution over the actions available to player p. **Behavioural strategies** have no constraints on the form of the action distributions. **Pure**

strategies, on the other hand, have action distributions that put all of the probability mass on a single action for every information set (*i.e.* a deterministic strategy).

Agents receive rewards at the end of a game based on the strategies they choose. We denote the amount of reward given to player p for reaching terminal node τ of the game tree as $V_p(\tau)$. The expected reward (over all terminal nodes) that each strategy receives is determined by a **payoff function** $\pi(\sigma_1, \ldots, \sigma_n) = (\pi_1(\sigma_1, \ldots, \sigma_n), \ldots, \pi_n(\sigma_1, \ldots, \sigma_n))$ which returns an n-tuple of the expected rewards for all players.

Now that we have defined some of the basic game theory terminology, we can define some more specific properties of games and strategies.

A **Nash equilibrium** is an n-tuple of strategies where none of the players can increase their payoff by only changing their own strategy. To be precise, an n-tuple of strategies $(\sigma_1^*, \ldots, \sigma_n^*)$ is a Nash equilibrium, if and only if, $\pi_p(\sigma_1^*, \ldots, \sigma_{p-1}^*, \sigma_p, \sigma_{p+1}^*, \ldots, \sigma_n^*) \leq \pi_p(\sigma_1^*, \ldots, \sigma_n^*)$ for all possible strategies σ_p of every player p. The payoff value of the equilibrium $\pi_p(\sigma_1^*, \ldots, \sigma_n^*)$ is often called the **value** of the equilibrium.

One class of games that has some interesting properties is two-player zero-sum games. In a **two-player zero-sum game** there are only two players and the sum of their rewards is zero, that is, $\pi_1(\sigma_1, \sigma_2) = -\pi_2(\sigma_1, \sigma_2)$. A **minimax optimal** strategy (usually just called an optimal strategy) in a two-player zero-sum game is a strategy that maximizes a player's *guaranteed* minimum expected reward under any possible opponent strategy. For two-player zero-sum games, Nash equilibria are optimal for both players because using a strategy that does not obtain the equilibrium value leaves the player vulnerable to exploitation by some strategy of the opponent. Moreover, if (σ_1^*, σ_2^*) and $(\hat{\sigma}_1^*, \hat{\sigma}_2^*)$ are both equilibria, then so is $(\sigma_1^*, \hat{\sigma}_2^*)$ and $(\hat{\sigma}_1^*, \sigma_2^*)$. Note that, in general, an optimal strategy does not necessarily *maximize* rewards.

A **dominated strategy** σ_p^* for player p is a strategy such that p has another strategy σ_p whose expected payoff is at least as much as σ_p^* against every possible opponent strategy and greater than σ_p^* for at least one possible opponent strategy. More specifically, σ_p^* is dominated if there exists σ_p such that $\pi_p(\sigma_1, \ldots, \sigma_{p-1}, \sigma_p^*, \sigma_{p+1}, \ldots, \sigma_n) \leq \pi_p(\sigma_1, \ldots, \sigma_n)$ for all σ_i where $i \neq p$ and $\pi_p(\sigma_1, \ldots, \sigma_{p-1}, \sigma_p^*, \sigma_{p+1}, \ldots, \sigma_n) < \pi_p(\sigma_1, \ldots, \sigma_n)$ for at least one choice of σ_i's where $i \neq p$.

Although there are Nash equilibrium solutions to many games, including Kuhn poker, our agent modelling work aims to maximize our rewards by learning an opponent's strategy and exploiting its weaknesses. To do this, we use the state estimation algorithms presented in the next section.

3.2 State Estimation

State estimation refers to the problem of determining the current state of a system given a sequence of our actions and a sequence of observations of the system. Formally, we represent the true state[1] at time t as the vector x_t. Similarly, let z_t and u_t denote vectors for the observation and action, re-

[1]Note that this is the true state of our *chosen* state parameters. This does not necessarily include all of a system's variables.

spectively, at time t. Define $x_{1:t}$ to be the state sequence $x_1, \ldots, x_t$ and similarly for the observation sequence $z_{1:t}$ and action sequence $u_{1:t}$. Then the state estimation problem is concerned with estimating the random variable $x_t|(z_{1:t}, u_{1:t})$, or rather the complete posterior distribution $\Pr(x_t|z_{1:t}, u_{1:t})$. Our presentation of state estimation algorithms begins with a description of Bayesian filters, followed by two specific Bayesian filtering algorithms: particle filters and Rao-Blackwellized particle filters. Note that much of the formalism presented in this section mirrors that of Thrun, Burgard, and Fox [25].

3.2.1 Bayesian Filtering

A common approach to state estimation is to represent the uncertainty in the current state as a probability distribution and use Bayes' rule to update the belief after every action/observation pair. This technique is called *Bayesian filtering*. By applying Bayes' rule and assuming that x_t is a sufficient statistic for the observations and actions up to time t (*i.e.* the Markov assumption), we arrive at the standard recursive Bayesian filtering equation.

$$\begin{aligned}
\Pr(x_t|z_{1:t}, u_{1:t}) &= \frac{\Pr(z_t|x_t, z_{1:t-1}, u_{1:t}) \Pr(x_t|z_{1:t-1}, u_{1:t})}{\Pr(z_t|z_{1:t-1}, u_{1:t})} && (3.1) \\
&= \eta \Pr(z_t|x_t, z_{1:t-1}, u_{1:t}) \Pr(x_t|z_{1:t-1}, u_{1:t}) && (3.2) \\
&= \eta \Pr(z_t|x_t) \Pr(x_t|z_{1:t-1}, u_{1:t}) && (3.3) \\
&= \eta \Pr(z_t|x_t) \int \Pr(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) \Pr(x_{t-1}|z_{1:t-1}, u_{1:t}) dx_{t-1} && (3.4) \\
&= \eta \Pr(z_t|x_t) \int \Pr(x_t|x_{t-1}, u_t) \Pr(x_{t-1}|z_{1:t-1}, u_{1:t-1}) dx_{t-1} && (3.5)
\end{aligned}$$

where η is the denominator in Equation 3.1, *i.e.*, the normalization constant. Equation 3.1 is derived by applying Bayes' rule. Equation 3.3 and Equation 3.5 follow from the Markov assumption. Marginalization over x_{t-1} yields Equation 3.4. Given our previous belief, $bel(x_{t-1}) \equiv \Pr(x_{t-1}|z_{1:t-1}, u_{1:t-1})$, we can use Equation 3.5 to find our new belief after the latest action and observation. Algorithm 1 provides pseudocode for the update.

Algorithm 1 Bayesian Filter

```
Require: $bel(x_{t-1}), u_t, z_t$
  for all $x_t$ do
    $\overline{bel}(x_t) = \int \Pr(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$
    $bel(x_t) = \eta \Pr(z_t|x_t) \overline{bel}(x_t)$
  Return $bel(x_t)$
```

The algorithm requires an *observation model* $\Pr(z_t|x_t)$, a *motion model* $\Pr(x_t|u_t, x_{t-1})$, and an initial belief or *prior* $\Pr(x_0)$. For a practical implementation, the form of the belief distribution $bel(x_t)$, the motion model, and the observation model all need to allow the integral in the Bayesian filtering equation to be computed easily. For example, using a Gaussian form for these distributions results in one of the Kalman filter variants. A Monte Carlo approximation results in a particle filter, which is the approach taken in this thesis.

3.2.2 Particle Filtering

A particle filter is a Bayesian filter that uses Monte Carlo methods with importance sampling. Particle filters approximate the probability distribution over the state using a set of samples called *particles*. Each particle is a state vector denoted as $x_t^{(i)}$. Updating a particle filter relies on importance sampling. To explain particle filters, we will first cover the basic idea behind importance sampling before elaborating on the implementation details of particle filtering.

Importance Sampling

Importance sampling is a general technique to estimate $\mathrm{E}[f(Q)]$ without having to actually sample from $Q \sim \Pr(Q = q)$. If we want an unbiased way to estimate $\mathrm{E}[f(Q)]$, but we can only sample from another distribution $\tilde{Q} \sim \Pr(\tilde{Q} = q)$ (referred to as a *proposal distribution*), then just sample from $\tilde{Q}$ and reweight each sample by the *importance weight* $w(q)$. This technique works because for any function f applied to Q we notice the following:

$$\begin{aligned}
\mathrm{E}[f(Q)] &= \int_q f(q)\Pr(Q = q)dq \\
&= \int_q f(q)\Pr(Q = q)\frac{\Pr(\tilde{Q} = q)}{\Pr(\tilde{Q} = q)}dq \\
&= \int_q f(q)\Pr(\tilde{Q} = q)w(q)dq \qquad \text{where } w(q) = \frac{\Pr(Q = q)}{\Pr(\tilde{Q} = q)} \\
&= \mathrm{E}[g(\tilde{Q})] \qquad \text{where } g(q) = w(q)f(q)
\end{aligned}$$

Therefore sampling from $\tilde{Q}$, with the importance weights $w(q)$, will have the same expectation as sampling from Q.

Implementation

We now describe the implementation of particle filtering and how the algorithm uses importance sampling. Returning to our Bayesian filtering problem, we want to sample $x_t \sim \Pr(x_t|z_{1:t}, u_{1:t})$. It is not currently known how to directly sample this distribution. Instead of sampling it directly we will use importance sampling. Let $\Pr(Q = q) \equiv \Pr(x_t|z_{1:t}, u_{1:t}) = \eta \Pr(z_t|x_t)\Pr(x_t|z_{1:t-1}, u_{1:t})$ (by Equation 3.3). Now we need an appropriate proposal distribution, $\Pr(\tilde{Q} = q)$. Both $\Pr(z_t|x_t)$ and $\Pr(x_t|z_{1:t-1}, u_{1:t})$ could be used since using one for the proposal distribution results in the other being used as the importance weight. In most particle filter applications the observation model $\Pr(z_t|x_t)$ is used for importance weights and the proposal distribution comes from the motion model $\Pr(x_t|z_{1:t-1}, u_{1:t})$ and the belief distribution $\Pr(x_{t-1}|z_{1:t-1}, u_{1:t-1})$. That being said, Thrun and colleagues have shown that using the observation model as the proposal distribution and the motion model and belief distribution for importance weights (a technique they call *dual Monte Carlo localization*) is also a viable approach despite some challenges in implementation [26]. Moreover, they present a technique to combine both standard and dual Monte Carlo localization in their *mixture Monte Carlo localization* algorithm. We use the first (standard) approach in this work.

Now that samples can be drawn from our desired distribution we can apply Monte Carlo methods. If we have a particle $x_{t-1}^{(i)}$ approximately sampled from $\Pr(x_{t-1}|z_{1:t-1}, u_{1:t-1})$ we want a new particle $x_t^{(i)}$ approximately sampled from $\Pr(x_t|z_{1:t}, u_{1:t})$. We do this by sampling $\tilde{x}_t^{(i)}$ from the (motion model) proposal distribution $\Pr(x_t|u_t, x_{t-1}^{(i)})$ and weighting it by the (observation model) importance sampling correction $\Pr(z_t|\tilde{x}_t^{(i)})$.

The final stage of particle filtering is *resampling*. Resampling is used to avoid degeneracy in the Monte Carlo sampling approximation where most of the particles begin to have (near) zero probability. In general, resampling randomly selects n particles from the previous set of Monte Carlo samples. There are numerous resampling algorithms, each with different computational costs, benefits (*e.g.* lower variance), and disadvantages (*e.g.* bias). Some examples of resampling algorithms include *low variance sampling* and *stratified sampling* [25]. The most straightforward resampling algorithm is to simply draw the n particles, with replacement, from the previous set, with probability proportional to their importance weighting. This resampling algorithm is the one we used in our experiments. Resampling has the benefit of focusing the limited number of particles in areas of higher probability. This allows the algorithm to discard unlikely particles for ones that are more "interesting". The full particle filtering algorithm can be seen in Algorithm 2.

Algorithm 2 Particle Filter

```
Require: A set of weighted particles $X_{t-1}$
Require: Action $u_t$ and observation $z_t$
  $\overline{X}_t = X_t = \emptyset$
  for $i = 1$ to $|X_{t-1}|$ do
    sample $x_t^{(i)} \sim \Pr(x_t|u_t, x_{t-1}^{(i)})$
    $w_t^{(i)} = \Pr(z_t|x_t^{(i)})$
    $\overline{X}_t = \overline{X}_t \cup \{\langle x_t^{(i)}, w_t^{(i)} \rangle\}$
  for $i = 1$ to $|X_{t-1}|$ do
    add $x_t^{(i)}$ to $X_t$ with probability $\propto w_t^{(i)}$
  Return $X_t$
```

Monte Carlo methods give particle filters considerable power and flexibility. They can handle non-linear dynamics while representing arbitrary belief distributions over the state variables. This is in contrast to many other Bayesian filtering algorithms, such as the Kalman filter variants, that are only capable of representing Gaussian probability distributions. Particle filters have one major drawback – they can be expensive. A particle filter's accuracy and computational cost scales with the number of particles used. Unfortunately, the number of particles needed to cover an n-dimensional state space is exponential in n. On the bright side, the number of particles to use is a simple parameter to manipulate. This means that the accuracy of a particle filter can easily scale as computer hardware improves.

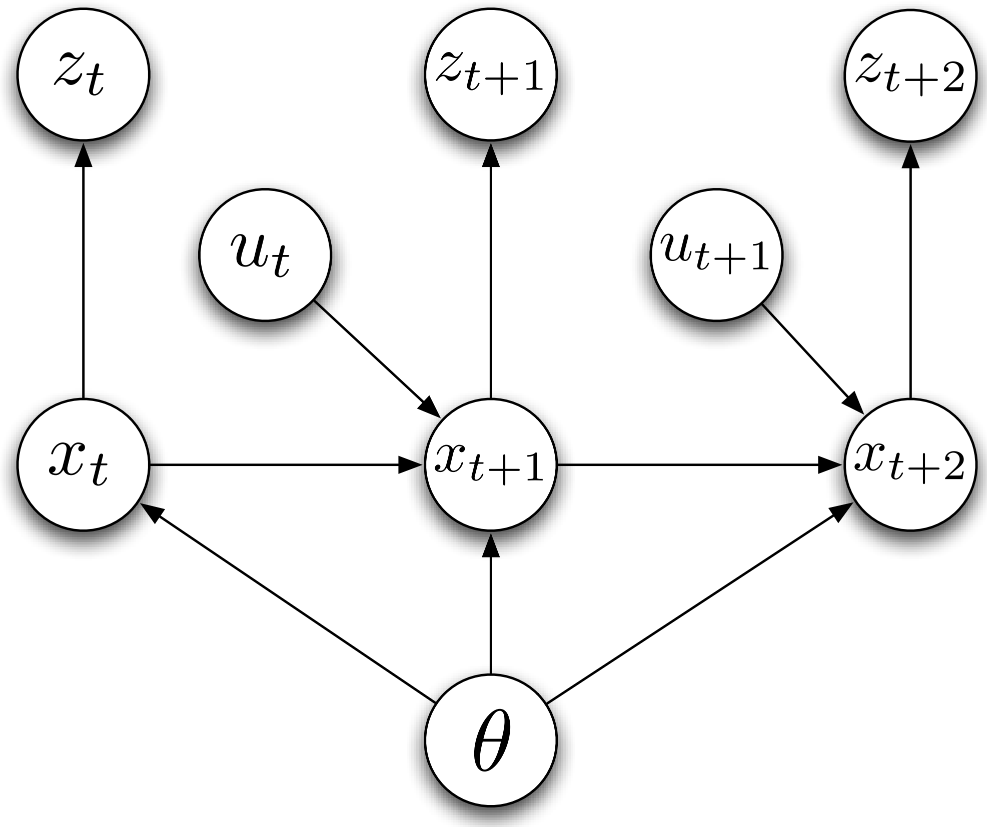


Figure 3.1: Graphical model of a dynamic agent's state

3.2.3 Rao-Blackwellized Particle Filtering

Rao-Blackwellized particle filters (RBPFs) are a hybrid Bayesian filtering technique that extend standard particle filters. More precisely, RBPFs are useful for estimating $E[f(X)]$ when X can be factored into a joint product of two sets of variables: one set X' that must be sampled, and another set X'' whose expectation can be integrated out analytically given X'. As one might guess, particle filters can be used to estimate X'. This hybrid inference, estimating both analytically and by sampling, is similar in many regards to standard particle filters. Like particle filters, RBPFs can handle non-linear dynamics and arbitrary belief distributions. Beliefs are still represented with a set of particles. The key difference lies in being able to factor X and analytically estimate X''.

In this work, we use RBPFs to perform *dual estimation*, inferring the system state and any unknown parameters of the motion or observation models simultaneously. We consider a simple case of dual estimation where the system's dynamics are parameterized by some unknown value θ, which we need to simultaneously infer along with the system state [17, 24]. Although RBPFs can be used more generally than this, we will only derive RBPFs that infer the hidden state and the motion model parameters as this is all that we need. Consider the general statistical process shown in Figure 3.1 that underlies this inference problem. The main difference between this process and the process for the Bayesian filter derived earlier is that the hidden state now depends on the motion model parameter θ. We begin the derivation of the RBPF posterior in much the same way as the Bayesian filter derivation.

$$\begin{aligned}
\Pr(x_{1:t}|z_{1:t}, u_{1:t}) &= \eta \Pr(z_t|x_{1:t}, z_{1:t-1}, u_{1:t}) \Pr(x_{1:t}|z_{1:t-1}, u_{1:t}) && (3.6) \\
&= \eta \Pr(z_t|x_t) \Pr(x_{1:t}|z_{1:t-1}, u_{1:t}) && (3.7) \\
&= \eta \Pr(z_t|x_t) \int_{x_{1:t-1}} \Pr(x_{1:t-1}|z_{1:t-1}, u_{1:t-1}) \\
&\qquad \Pr(x_t|x_{1:t-1}, z_{1:t-1}, u_{1:t}) && (3.8)
\end{aligned}$$

where η is a normalization constant. Equation 3.6 is derived by applying Bayes' rule. Equation 3.7 follows from z_t being conditionally independent of the other variables given x_t (see Figure 3.1). Marginalization over the chain of hidden states $x_{1:t-1}$ and the fact that $x_{1:t-1}$ does not depend on u_t yields Equation 3.8. Equation 3.8 still presents us with a problem since we do not know how to compute $\Pr(x_t|x_{1:t-1}, z_{1:t-1}, u_{1:t})$ due to the dependency of x_t on the unknown value of θ. To solve this problem we marginalize over θ.

$$\begin{aligned}
\Pr(x_{1:t}|z_{1:t}, u_{1:t}) &= \eta \Pr(z_t|x_t) \int_{x_{1:t-1}} \Pr(x_{1:t-1}|z_{1:t-1}, u_{1:t-1}) \\
&\qquad \int_{\theta} \Pr(x_t|x_{1:t-1}, z_{1:t-1}, u_{1:t}, \theta) \Pr(\theta|x_{1:t-1}, z_{1:t-1}, u_{1:t}) && (3.9) \\
&= \eta \Pr(z_t|x_t) \int_{x_{1:t-1}} \Pr(x_{1:t-1}|z_{1:t-1}, u_{1:t-1}) \\
&\qquad \int_{\theta} \Pr(x_t|x_{t-1}, u_t, \theta) \Pr(\theta|x_{1:t-1}, z_{1:t-1}, u_{1:t-1}) && (3.10)
\end{aligned}$$

Equation 3.10 follows from Equation 3.9 for two reasons. First, x_t is conditionally independent of the other variables given x_{t-1}, u_t, and θ. Second, $\Pr(\theta|x_{1:t-1}, z_{1:t-1}, u_{1:t})$ is independent of u_t given $x_{1:t-1}, z_{1:t-1}, u_{1:t-1}$. Equation 3.10 gives us an expression for our posterior distribution that is similar to the posterior for our Bayesian filter (Equation 3.5). Like before we have a recursive update for our belief distribution $\Pr(x_{1:t-1}|z_{1:t-1}, u_{1:t-1})$ and we require an observation model $\Pr(z_t|x_t)$, and a motion model $\Pr(x_t|x_{t-1}, u_t, \theta)$ (which is now dependent on θ). This leaves us with the one new term $\Pr(\theta|x_{1:t-1}, z_{1:t-1}, u_{1:t-1})$ that we now consider how to compute.

$$\begin{aligned}
\Pr(\theta|x_{1:t}, z_{1:t}, u_{1:t}) &= \eta \Pr(x_t, z_t|x_{1:t-1}, z_{1:t-1}, u_{1:t}, \theta) \Pr(\theta|x_{1:t-1}, z_{1:t-1}, u_{1:t}) && (3.11) \\
&= \eta \Pr(z_t|x_t) \Pr(x_t|x_{1:t-1}, z_{1:t-1}, u_{1:t}, \theta) \\
&\qquad \Pr(\theta|x_{1:t-1}, z_{1:t-1}, u_{1:t}) && (3.12) \\
&= \eta' \Pr(x_t|x_{t-1}, u_t, \theta) \Pr(\theta|x_{1:t-1}, z_{1:t-1}, u_{1:t-1}) && (3.13)
\end{aligned}$$

where η is a normalization constant from Bayes' rule. Equation 3.12 follows from the multiplication rule of conditional probabilities and the fact that z_t is is independent from the other variables given x_t. Since $\Pr(z_t|x_t)$ is not dependent on θ we can move it into the normalization constant making $\eta' = \eta \Pr(z_t|x_t)$. The new normalization constant combined with the observations from the derivation of Equation 3.10 yields the recursive update shown in Equation 3.13.

If we choose a distribution for $\Pr(x_t|x_{t-1}, u_t, \theta)$ that has a conjugate prior, then our posterior at time t will have the same form as the posterior at time $t-1$. This means that we only need to store and update a sufficient statistic s_t to compute $\Pr(\theta|x_{1:t}, z_{1:t}, u_{1:t})$. More precisely, we can let

$$\Pr(\theta|x_{1:t}, z_{1:t}, u_{1:t}) \quad = \quad \Pr(\theta|s_t) \tag{3.14}$$

By using a conjugate prior, the recursive structure of Equation 3.13 enables us to compute our posterior $\Pr(\theta|x_{1:t}, z_{1:t}, u_{1:t})$ efficiently, incurring relatively little extra computational cost compared to basic particle filters.

Using our derivations we can augment our particle filter algorithm (Algorithm 2) to model a process with unknown motion model parameters. We present the augmented particle filter, which is just a specific case of a Rao-Blackwellized particle filter, in Algorithm 3. Each particle stores the additional sufficient statistic, $s_t^{(i)}$ for $\Pr(\theta|s_t^{(i)})$. Sampling from the proposal distribution now involves sampling $\tilde{\theta}$ from $\Pr(\theta|s_{t-1}^{(i)})$ and then sampling $\tilde{x}_t^{(i)}$ from $\Pr(x_t|u_t, x_{t-1}^{(i)}, \tilde{\theta})$. The sufficient statistic for each candidate particle is updated for the new transition $s_t^{(i)} = \textsc{Update}(s_{t-1}^{(i)}, x_{t-1}^{(i)} \rightarrow \tilde{x}_t^{(i)})$. The weighting from the observation model and resampling is performed in the usual fashion.

Algorithm 3 Rao-Blackwellized Particle Filter

```
Require: A set of weighted particles $X_{t-1}$
Require: Action $u_t$ and observation $z_t$
  $\overline{X}_t = X_t = \emptyset$
  for $i = 1$ to $|X_{t-1}|$ do
    sample $\tilde{\theta} \sim \Pr(\theta|s_{t-1}^{(i)})$
    sample $x_t^{(i)} \sim \Pr(x_t|u_t, x_{t-1}^{(i)}, \tilde{\theta})$
    $s_t^{(i)} = \textsc{Update}(s_{t-1}^{(i)}, x_{t-1}^{(i)} \rightarrow x_t^{(i)})$
    $w_t^{(i)} = \Pr(z_t|x_t^{(i)})$
    $\overline{X}_t = \overline{X}_t \cup \{\langle x_t^{(i)}, s_t^{(i)}, w_t^{(i)} \rangle\}$
  for $i = 1$ to $|X_{t-1}|$ do
    add $\langle x_t^{(i)}, s_t^{(i)} \rangle$ to $X_t$ with probability $\propto w_t^{(i)}$
  Return $X_t$
```

3.2.4 Related Applications

State Estimation algorithms have been applied in many domains to solve a diverse range of problems. Particle filters have been used in vision systems under the guise of the condensation algorithm for detecting and tracking the contours of moving objects [14]. In robotics, state estimation algorithms are used extensively for robotic localization where a robot's pose must be inferred from the robot's actions (movement) and observations (sensor data) [25].

More recently, the problem of simultaneous localization and mapping (SLAM) has been examined in robotics research. The goal of SLAM, as its name suggests, is to localize a robot while simultaneously building a map of the robot's environment. Work by Montemerlo and colleagues presented a new approach to SLAM, called FastSLAM, that uses RBPFs [18]. Their work assumes that measurements of individual landmarks in the environment are conditionally independent given the robot's path. This allows them to factor the SLAM problem into an estimation of the sufficient statistics ($s_t^{(i)}$) for the landmark locations and an estimation of the robot's pose ($x_t^{(i)}$). By inferring the location of 50,000 landmarks, their experiments showed that RBPFs are both effective and efficient for solving the SLAM problem when this assumption holds.

State estimation has also been proposed for giving computer agents a more realistic belief about opponent positions in commercial games. More precisely, Bererton proposed the use of particle filters to track an agent's (x, y) location in a world with several rooms [2]. A simple motion model using a random walk is used to represent the opponent's movements. Particles are weighted based on two cases. If the opponent is visible a particle is weighted according to its distance from the opponent, otherwise the agent simulates a laser range finder sweep and particles are weighted based on their distance from a laser ray.

Although these are all interesting applications, we are interested in using state estimation for inferring an agent's subjective state, *i.e.* its behaviour, rather than an objective state like the position of an agent or an object. The next chapter describes our application of state estimation to the problem of inferring an agent's strategy in the domain of Kuhn poker.

Chapter 4

Application to Kuhn Poker

We now describe our application of state estimation techniques, specifically particle filters and Rao-Blackwellized particle filters, to the game of Kuhn poker. First, we will discuss Kuhn poker in further depth including a description of *undominated* Kuhn poker. Next, we will describe how we mapped the structure of Kuhn poker onto Bayesian filtering, including some elaboration on our choice of motion models. We will then describe the extra requirements for Rao-Blackwellized particle filters. Finally, we explain how we use the opponent model when playing.

4.1 Undominated Kuhn Poker

Section 2.5.1 introduced the game of Kuhn poker: a two player zero-sum poker game where each player is dealt a single card from a three card deck. In addition to defining the game, Kuhn also presented a complete game theoretic analysis [16]. Although there are 64 different pure strategies for each player, many of these are dominated. By removing all of the dominated strategies, we are left with the game of *undominated* Kuhn poker. Figure 4.1 shows the game tree for undominated Kuhn poker. For the remainder of this thesis, we will be working in the domain of undominated Kuhn poker which we refer to as simply "Kuhn poker". To simplify things further, we restrict ourselves to the situation where we are player one modelling player two.

In undominated Kuhn poker, the strategy space of player one can be parameterized by three parameters (α, β, γ), and player two by two parameters (η, ξ). These parameters are all in the range $[0, 1]$ and specify the probability of betting in certain information sets. For example, η is the probability the second player bets when facing a bet while holding the queen, and ξ is the probability the second player bets after a pass when holding the jack.

Kuhn poker has a continuum of Nash equilibria, which can be written in the aforementioned parameterization as, $\alpha = \gamma/3$, $\beta = (1 + \gamma)/3$, and $\eta = \xi = 1/3$. The value of the equilibrium is $-1/18$ dollars per game. In other words, if player one plays an equilibrium strategy then they can expect to lose at most 5.5 cents per game. Similarly, if player two plays an equilibrium strategy then they will win at least 5.5 cents per game on average.

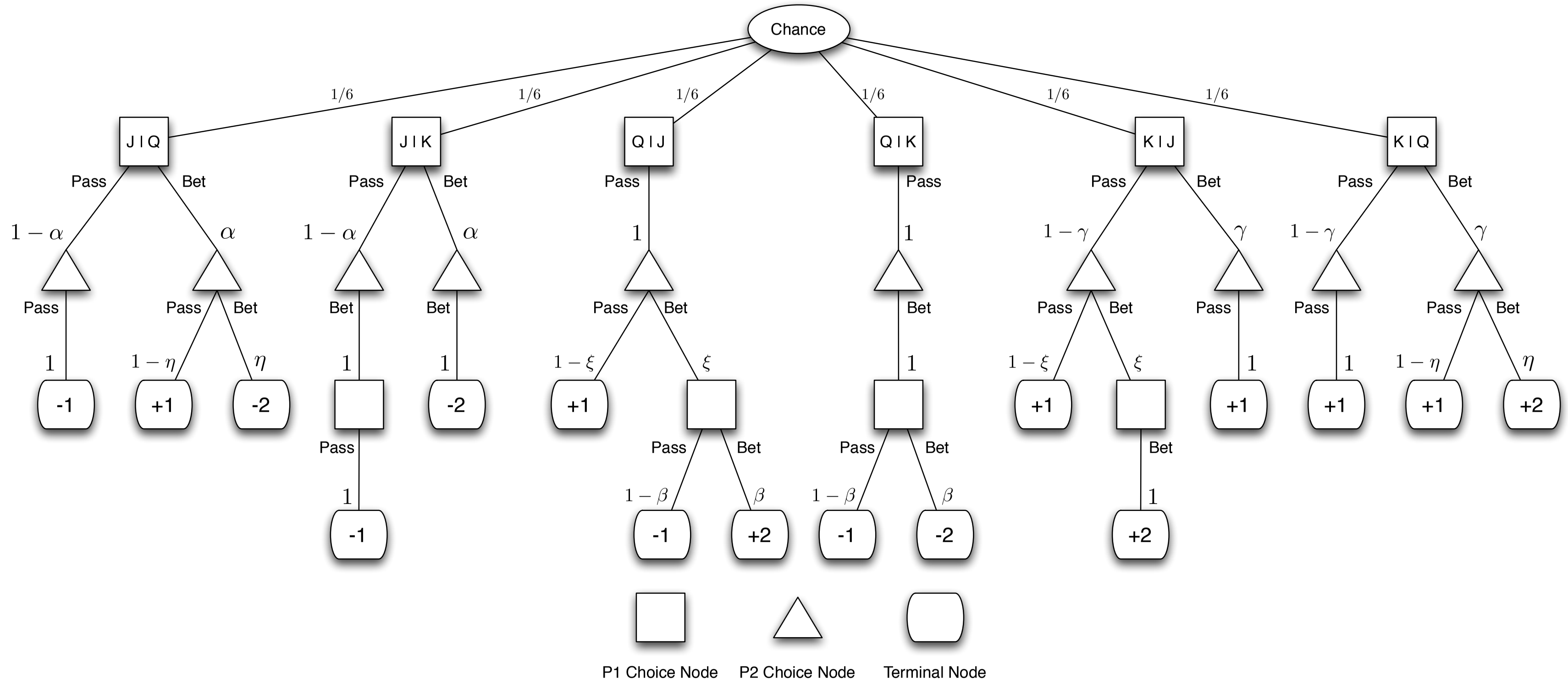


Figure 4.1: The full game tree of undominated Kuhn poker

Although many of the competitive poker programs currently employ Nash equilibrium approximations [3, 10], greater payoffs may still be possible. Consider the following example from Kuhn poker. Suppose player two chooses undominated actions with equal probability (*i.e.* $\eta = \xi = 0.5$). If player one plays an equilibrium strategy, they can expect to lose $-1/18$ per game. Now suppose player one deviates from the equilibrium and responds by passing in the first round and betting with a king or queen when bet to (*i.e.* the best response). This is no longer a losing game for player one who now has an expected payoff of zero. Other deviations from the equilibrium strategy can be exploited by even more depending on player two's choice of strategy. In summary, an accurate model of the opponent's strategy can be used to great advantage.

4.2 Bayes Filter Components

To use particle filtering, or any Bayesian filtering algorithm, we need to define six components: the state variables, observations, actions, an observation model, a motion model, and an initial belief.

4.2.1 Observations

In general, an agent receives observations each time they arrive at an information set, *i.e.*, whenever new public or private information is revealed. In our domain of Kuhn poker, this occurs after each decision is made and whenever cards are revealed. Instead of using each information set as an observation, we use a full hand of the game as the observation for each time step in the Bayesian filter update. We use this approach for two reasons: the simplicity of the resulting filter, and the fact that player two only makes one decision per hand. With the filter being updated after each hand, an observation z_t corresponds to the information set for player one at a terminal node in the game tree. This means that the observation will consist of the betting sequence for the hand and any known cards held by each player. We will denote a Kuhn poker information set from player p's perspective as $I_p = \langle H_1\, H_2\ :\ D\rangle$ where H_i is the cards held by player i and D is the public betting information.

For example, the information set $\langle J\, Q\ :\ Bb\rangle$ would be observed when player one has the jack, player two has the queen, and player one bet followed by a bet from player two. Note that at the end of a hand we may not know what cards the other player holds. An example of this is when player one is dealt a jack and bets followed by player two passing. In this case, the information set would be $\langle J\, ?\ :\ Bp\rangle$ with the question mark denoting unknown information.

4.2.2 Observation Model

A general observation model $\Pr(z_t|x_t)$ must be able to compute the probability of an observation given the current beliefs about the system being modelled. In Kuhn poker, an observation model represents the probability of the betting and cards that were observed on this hand, given our current beliefs about the opponent. In order to compute this probability accurately for our case of player one modelling player two, our strategy parameters x_t will need to contain, or from it be able to derive, the betting probabilities η_t and ξ_t from Figure 4.1. Given these parameters, the observation model $\Pr(z_t|\eta_t, \xi_t)$ comes from the definition of the game itself. The observation model then returns the product of the probability of the cards that were dealt, the probabilities for the modelling agent's actions, and the probabilities for the opponent's observed decisions given η_t and ξ_t.

Note that the observation model can often omit terms whose probability is the same regardless of the current beliefs. For instance, our observation model does not need to account for the probabilities of the modelling agent's actions since they are constant for all values of η_t and ξ_t. The same is true for the probability of the cards that were dealt because all of the chance event outcomes have the same probability. Keep in mind that omitting these terms is only possible when the observation model can be normalized after the fact or when the Bayesian filtering algorithm being used does not require normalized probabilities (as is the case with particle filters).

To illustrate our observation model, consider the case where we observe the information set $\langle K\,J\ :\ PbB\rangle$. Then the observation model should return the product of the probabilities for player two's decisions in the branch with the terminal node corresponding to $\langle K\,J\ :\ PbB\rangle$. This works out to ξ_t. If, instead, the information set was $\langle K\,J\ :\ Pp\rangle$, then the unnormalized probability would be $(1 - \xi_t)$.

Now we examine the case where an opponent's cards are not known. Consider the information sets $\langle K\,J\ :\ Bp\rangle$ and $\langle K\,Q\ :\ Bp\rangle$. Since the second player passes, player one never observes their card. From player one's perspective, both of these information sets "look" the same. In this case player one observes $\langle K\,?\ :\ Bp\rangle$. When an observation has unknown cards the observation model can compute the probability by marginalizing over the unknown cards. Specifically, when the observed information set has unknown cards ($H_2\ = ?$) the observation model returns $\sum_{H_2} \Pr(z_t|H_2, \eta_t, \xi_t)$. In our example, summing the probability of each of the possible branches yields an unnormalized probability of $(1 + (1 - \eta))$.

4.2.3 State Variables

As we mentioned in Section 4.2.2, the simplest possible state x_t that we can have while providing enough information to accurately compute our observation model is to let $x_t = [\eta_t, \xi_t]$. We chose to use this minimal representation since it is both simple and it was also used by Hoehn and colleagues [12, 13]. This choice makes two assumptions. First, that our opponent will not play outside the parameterization (*i.e.* does not play dominated strategies). Second, our opponent's future strategies are conditionally independent of past strategies given their current strategy (*i.e.* the Markov property). These assumptions create some limitations on the type of behaviour dynamics that we can model, but this parameterization is sufficient for the motion models that we have opted to use. These models are discussed in the next section.

4.2.4 Motion Models

In general, motion models encode our belief about the state dynamics of the system being modelled. In our case, motion models are used to encode how other agents will change their strategies over time. In this thesis we explore two "naive" types of motion models. More precisely, these "naive" motion models assume that our opponent is **oblivious**, *i.e.*, they ignore past observations of an agent's behaviour when making their current decision. This means that the motion models assume that the opponent's strategy is independent of the modelling agent's past actions.

Our first model assumes that players will change with probability ρ to a uniform random strategy after every hand. With probability $(1 - \rho)$ they continue using their previous strategy. We call this a *switching* model. The switching parameter ρ fully describes the dynamics in this model.

Our second model, which we call a *drifting* model, assumes that players' strategies drift after each hand. Specifically, a player generates each of their next strategy parameters by independently sampling from a Gaussian distribution with a mean of the current strategy parameter (*i.e.* η or ξ) and standard deviation σ. We reject any parameter sampled outside of $[0, 1]$. The resulting parameter is therefore a sample from a truncated Gaussian distribution. The standard deviation σ is the single parameter that describes the dynamics in this model.

Finally, we also refer to a combined motion model which involves both the ρ and σ parameters. In this model, the player switches to a uniform random strategy at the end of the hand with probability ρ and with probability $(1 - \rho)$ it drifts with standard deviation σ.

Now that we have parameterized models of agent dynamics, the parameters must be given a value. The simplest way to do this is for a human to choose parameter values that are believed to be appropriate. The motion model and assigned parameter value could then be used as the motion model component of a particle filter. Alternatively, these parameters could be inferred using the RBPF algorithm described in Section 3.2.3. This approach requires a few extra components to be defined and is discussed further in Section 4.3.

4.2.5 Initial Belief

As was shown in Algorithm 1, Bayesian filters require a belief distribution over the previous time step, $bel(x_{t-1})$, to estimate the current beliefs, $bel(x_t)$. This means that we need to define a probability distribution for the modelling agent's beliefs at time step zero (*i.e.* $bel(x_0)$). An accurate initial belief is always desirable since it gives us an accurate model to begin prediction with.

We decided to use a uniform random distribution over the state space because it is both simple and requires no additional knowledge of the opponent. A more informed initial belief would be beneficial if more is known a priori. For example, one possible "informed" initial belief would be to have a higher probability that our opponent would choose a strategy close to the equilibrium. Of course the effectiveness of an initial belief depends on how well it corresponds to the actual initial strategies one's opponents employ.

4.2.6 Actions

Because our naive motion models do not account for the opponent modelling us in any way, we do not need to incorporate our actions into our Bayesian filtering updates. The math of Equation 3.5 still holds if we remove the actions u_t. Another way to think about it is that we do have actions but they do not affect the motion model and therefore it does not matter what value they have. If we wanted more elaborate motion models that attempt to reason about how our opponent is modelling us, then we would need to add proper actions into the model. These actions would consist of the betting that happened during the hand and our cards if revealed.

4.3 Rao-Blackwellized Particle Filter Components

As we have chosen motion models that are parameterized, we can employ an RBPF to infer our motion model parameters, ρ and σ. Two more components need to be addressed to use the RBPF: the sufficient statistics of the conjugate priors for our motion model parameters (ρ, σ), and an initial belief for our sufficient statistics.

4.3.1 Sufficient Statistics

We briefly mentioned in Section 3.2.3 that RBPFs add little extra computational cost above regular particle filters as long as we choose a distribution for $\Pr(x_t|x_{t-1}, u_t, \theta)$ that has a conjugate prior. This allows us to learn and compute $\Pr(\theta|x_{1:t}, z_{1:t}, u_{1:t})$ using only the sufficient statistics s_t of our conjugate prior. We now describe the sufficient statistics of the conjugate priors used to represent the modelling agent's beliefs about its opponent's dynamics.

The binary outcomes of our switching model make a beta distribution, $\textsc{Beta}(a, b)$, ideal for our conjugate prior over ρ. More formally, if X is drawn from a beta distribution with parameters $a > 0$ and $b > 0$, then X has the following probability density function f and mean $E[X]$:

$$\begin{aligned} f(X = x; a, b) &= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1} \\ &= \frac{x^{a-1}(1-x)^{b-1}}{\mathrm{B}(a,b)} \\ \mathrm{E}[X] &= \frac{a}{a+b} \end{aligned}$$

where $\mathrm{B}(a, b)$ is the beta function. Using a beta distribution means that a and b are the sufficient statistics for ρ. Upon observing a transition the sufficient statistics are updated as follows: if $x_{t-1} = x_t$ increment b, otherwise increment a.

Inverse-gamma distributions (or inverse-Wishart distributions in the multivariate case) have often been used as conjugate priors for normal distributions. This work also uses them for this purpose and represents the distribution over the variance σ^2 of the drifting model's Gaussian distribution as an inverse-gamma, $\textsc{Inv-Gamma}(v, w)$. Formally, if X is drawn from an inverse-gamma distribution with shape parameter $v > 0$ and scale parameter $w > 0$ then X has the following probability density function f and mean $E[X]$:

$$\begin{aligned} f(X = x; v, w) &= \frac{w^v}{\Gamma(v)} x^{-v-1} \exp\left(\frac{-w}{x}\right) \\ \mathrm{E}[X] &= \frac{w}{v-1} \qquad \text{for } v > 1 \end{aligned}$$

Using an inverse-gamma distribution results in v and w being the sufficient statistics for σ^2. In general, to update v and w to account for a new observation x of the Gaussian (with known mean μ), we let $v = v + 1/2$ and $w = w + (x - \mu)^2/2$. Because we independently sample the Gaussian for each of our parameters, transitions are treated as one observation of σ^2 per dimension. On observing a transition the update adds $||x_t - x_{t-1}||^2/2$ to w and $d/2$ to v, where d is the number of dimensions in the state vector (two, in our case).

Using these sufficient statistics in Algorithm 3 consists of storing the sufficient statistics with each particle's state and updating them on each time step according to the aforementioned rules. Our sufficient statistic for the combined model for particle i at time step t is $s_t^{(i)} = [a, b, v, w]$. Of course if we wanted to change our modelling agent's motion models, $s_t^{(i)}$ would need to represent all of the sufficient statistics necessary for the new motion models.

4.3.2 Sufficient Statistic Priors

Finally, we need to choose a prior belief over ρ and σ. For our experiments we tried to choose prior distributions that looked "reasonable". We fairly arbitrarily settled on $\rho \sim \textsc{Beta}(1, 30)$ and $\sigma^2 \sim \textsc{Inv-Gamma}(0.6, 0.00005)$. Note that our inverse-gamma prior has an undefined mean since, for $v = 0.6 \leq 1$, the integral for the expectation of the inverse-gamma is infinite.

4.4 Using Our Model

We now consider what to do with our posterior belief about the opponent's strategy. The correct Bayesian approach is to select the action that maximizes the sum of all future expected utility given our belief. This utility is dependent on the information we learn about our opponent and the information our opponent learns about us. In general, this computation is intractable as it requires us to consider not only our own actions but how our actions impact our future winnings. Finding the Bayesian best-response would consist of running an expectimax algorithm on a lookahead tree of size $|I|^d = 16^d$ nodes where d is the number of hands left in our match and $|I|$ is the number of possible terminal information sets in a single hand (*i.e.* the number of leaves in Figure 4.1). For any reasonable length of game, this is not computable. An alternative to solving the full Bayesian best-response would be solving a Bayesian best-response that only uses a small lookahead. This would not guarantee the same degree of exploitation as the full best-response, but the limited size of the tree would make the computation tractable.

Because our motion models assume the opponent is oblivious and due to the computational cost in performing lookahead, we have opted to compute the *greedy Bayesian best response* rather than attempting any kind of lookahead. Specifically, our greedy Bayesian best response tries to maximize the value of the current hand (*i.e.* a lookahead of depth $d = 1$). This is the same approach used by Bayes' Bluff [23]. Computing this greedy response consists of choosing a set of parameters that will maximize the modelling agent's expected value on the current hand given its current beliefs. The greedy response in this case is to play the best-response to the mean of our particles.

To show this, we first need to define the expected value of a hand of Kuhn poker from player one's perspective given each player's strategy parameters. Examining the terminal information sets T of the game tree in Figure 4.1, we can determine an equation for this expected value.

$$\begin{aligned}
\mathrm{E}[I_1|\alpha,\beta,\gamma,\eta,\xi] &= \sum_{I_1 \in T} V_1(I_1)\cdot \Pr(I_1) \\
&= \frac{1}{6}\cdot\Big[-1\cdot(1-\alpha)+1\cdot(1-\eta)\alpha+-2\eta\alpha+-1\cdot(1-\alpha)+-2\alpha+ \\
&\qquad 1\cdot(1-\xi)+-1\cdot(1-\beta)\xi+2\beta\xi+-1\cdot(1-\beta)+-2\beta+ \\
&\qquad 1\cdot(1-\xi)(1-\gamma)+2\xi(1-\gamma)+1\gamma+1\cdot(1-\gamma)+ \\
&\qquad 1\cdot(1-\eta)\gamma+2\eta\gamma\Big] \\
&= \frac{1}{6}\cdot\big[\eta(\gamma-3\alpha)+\xi(3\beta-\gamma-1)+(\alpha-\beta)\big] \qquad (4.1)
\end{aligned}$$

If we hold player one's parameters (α,β,γ) constant in Equation 4.1, then the equation is linear in our chosen state variables η and ξ.

Next, we will use this result to show *that the best-response to our particle posterior is just the best-response to the mean of the particles*. At time step t, we want our modelling agent to choose strategy parameters $s = [\alpha\,\beta\,\gamma]$ that maximize the expected value over our opponent's unknown strategy parameters $o_t = [\eta_t\,\xi_t]$ given our beliefs about their previous strategy $o_{t-1} = [\eta_{t-1}\,\xi_{t-1}]$ and their motion model M.

$$\begin{aligned}
&\max_s \mathrm{E}\big[\mathrm{E}[I_1|s,o_t]|o_{t-1},M\big] \\
&= \max_s \mathrm{E}\left[\frac{1}{6}\cdot\big[\eta_t(\gamma-3\alpha)+\xi_t(3\beta-\gamma-1)+(\alpha-\beta)\big]|o_{t-1},M\right] \qquad (4.2) \\
&= \max_s \Big(\frac{1}{6}\cdot\big(\mathrm{E}[\eta_t(\gamma-3\alpha)|o_{t-1},M]+ \\
&\qquad \mathrm{E}[\xi_t(3\beta-\gamma-1)|o_{t-1},M]+\mathrm{E}[(\alpha-\beta)|o_{t-1},M]\big)\Big) \qquad (4.3) \\
&= \max_s \Big(\frac{1}{6}\cdot\big((\gamma-3\alpha)\cdot\mathrm{E}[\eta_t|o_{t-1},M]+ \\
&\qquad (3\beta-\gamma-1)\cdot\mathrm{E}[\xi_t|o_{t-1},M]+(\alpha-\beta)\big)\Big) \qquad (4.4) \\
&= \max_s \mathrm{E}\big[I_1|s,\mathrm{E}[o_t|o_{t-1},M]\big] \qquad (4.5)
\end{aligned}$$

Equation 4.2 substitutes Equation 4.1 into the expectation. Equation 4.3 follows from the linearity of expectations. Equation 4.4 results from α, β, and γ being fixed constants in the expectation. Finally, Equation 4.5 uses the equality from Equation 4.1.

Finding the strategy that maximizes Equation 4.5 (*i.e.* the best response strategy) can be done by differentiating Equation 4.1 with respect to α, β, and γ. Solving these derivatives for when they are positive gives us a set of inequalities for determining how to set the modelling agent's parameters to maximize Equation 4.5. These derivations are shown below.

$$\begin{aligned}
\frac{d}{d\alpha}\,\mathrm{E}[I_1|\alpha,\beta,\gamma,\eta,\xi] &= (1-3\eta)/6 > 0 &\iff&\quad \eta < 1/3\\
\frac{d}{d\beta}\,\mathrm{E}[I_1|\alpha,\beta,\gamma,\eta,\xi] &= (3\xi-1)/6 > 0 &\iff&\quad \xi > 1/3\\
\frac{d}{d\gamma}\,\mathrm{E}[I_1|\alpha,\beta,\gamma,\eta,\xi] &= (\eta-\xi)/6 > 0 &\iff&\quad \eta > \xi
\end{aligned}$$

Letting each parameter be 1 when its corresponding derivative is positive and 0 otherwise (recall that the values must be in $[0, 1]$) gives a parameter setting that maximizes the modelling agent's expected value on a given hand if their beliefs are true. For a hand at time t we compute and play the best-response to the mean of our particle posterior at time $t-1$.

Chapter 5

Experimental Results

This chapter evaluates our particle filter and RBPF modelling approaches against a variety of opponents, both static and dynamic, and when our prior beliefs are both correct and incorrect. This will help us to understand the effectiveness and robustness of these approaches. We begin our analysis by examining static opponents and then move on to dynamic opponents.

5.1 Static Opponents

Hoehn and colleagues' previous work on opponent modelling in Kuhn poker focused on static opponents [12, 13]. One of their experiments was designed to compare the quality of a number of different exploration strategies. These strategies were used during an initial exploration phase to help build an accurate opponent model. After exploration, their modelling agent would stop learning and switch to exploiting the learned model using the greedy best response we described in Section 4.4. The hand where the modelling agent switches to exploitation of its model is called the *switching hand*. We perform a similar experiment using our particle filtering approach with two of their balanced exploration strategies. Our experiment aims to answer three questions. Is our particle filtering approach comparable to, or even better than, Hoehn's parameter estimation when the opponent is static? How beneficial is it to continue learning after the exploration phase? Is an exploration phase needed if the modelling agent continues to learn throughout the interaction?

We play a particle filter modeller using a stationary motion model against Hoehn's six static opponents, O_1 through O_6. Each opponent corresponds to a different best response strategy for player one. An opponent's exact values for η and ξ are provided in the figures. For each opponent, we display results for two of Hoehn's exploration strategies both when we stop learning after the switching hand, and also when we continue to learn throughout the full length of the match. The exploration strategies we use are the Nash equilibrium strategy for player one with $\gamma = 0.75$ and the BalancedExplore strategy. This equilibrium strategy has strategy parameters of $[\alpha, \beta, \gamma] = [0.25, 0.583, 0.75]$. This strategy ensures η and ξ will have equal probability of being observed with complete information while achieving the equilibrium value. On the other hand, BalancedExplore uses $[\alpha, \beta, \gamma] = [1, 1, 0.5]$. Although this strategy is more vulnerable to exploitation, it avoids actions that prevent player one from learning about player two's parameters while ensuring η and ξ are observed with equal probability.

The resampling phase of the particle filter is disabled for these experiments because resampling when the particles are stationary is unnecessary and would cause particle impoverishment almost immediately. This topic will be covered in greater detail in the particle impoverishment segment of Section 5.2.1. Our results display each approach's expected total winnings at the end of a match. Each match consists of 200 hands between a particle filter of 1000 particles and one of the six static opponents. Matches are repeated 5000 times for statistical confidence.

We begin our analysis with an examination of Hoehn's parameter estimation results together with our particle filtering approach when learning is stopped. The results of this comparison are presented in Figures 5.1 through 5.3. These results show that our particle filtering approach, when learning is stopped after switching from exploration to exploitation, tends to benefit overall from the exploration phase. The amount of gain from exploration varies considerably depending on the opponent and the exploration strategy. The equilibrium exploration strategy yields the largest improvements when used for the first 30 to 50 hands (except against O_2 which did not improve at all). The BalancedExplore strategy, on the other hand, gives the largest performance improvements against O_1 around hand 20 and around hand 50 against O_4, O_5, and O_6. BalancedExplore provides no benefit against O_2 and O_3. More importantly, these results show that our particle filtering technique yields similar performance to Hoehn's parameter estimation. Note that our particle filtering results lack the smoothness of Hoehn's parameter estimation. This is due to the fact that our results were generated by actually playing the game whereas Hoehn's results were computed using a closed form calculation of the game's expected value. This adds noise to our computation, but our experiments yielded 95% confinence intervals that are no larger than ± 0.72.

Next, we present results of our particle filtering approach when learning is stopped and when learning continues throughout the match in Figures 5.4 through 5.6. These results allow us to compare the modeller's performance when learning is stopped at the switching hand versus continuing to learn throughout the match. As the switching hand increases, the exploration strategy is used for a longer portion of the match, so continued learning has less of an effect. On the other hand, when the switching hand is smaller, we consistently see that continuing to learn improves performance over stopping learning. One possible exception to this is O_6. In this case there are points where stopping learning appears beneficial when using the BalancedExplore strategy. A Wald test with 95% confidence at the switching hand where the difference is greatest indicates that this improvement is not statistically significant.

Focussing now on the case where learning continues throughout the match, we note that using a switching hand of 0 (immediately exploiting the prior) tends to outperform any kind of exploration. The exceptions to this are against opponents O_4 and O_6. In these cases, exploring with the

	Motion Model	
Opponent	Switch (ρ)	Drift (σ)
A	0.0	0.0
B	0.002	0.005
C	0.01	0.02
D	0.05	0.05
E	0.1	0.1

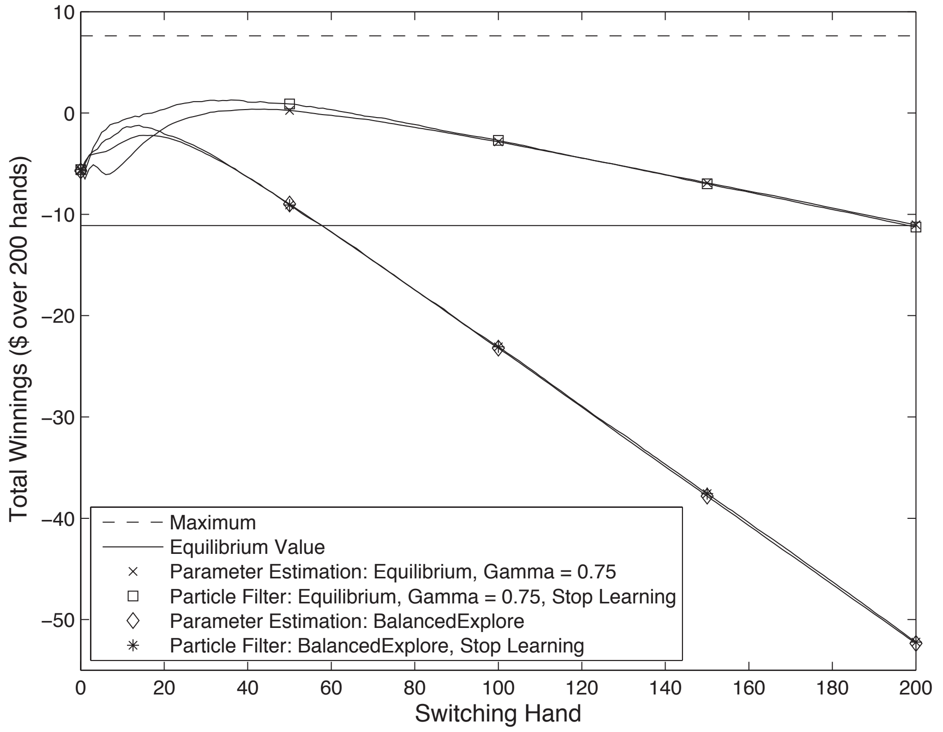
Table 5.1: Oblivious Dynamic Opponent Legend

BalancedExplore strategy provides a statistically significant improvement according to a Wald test with 95% confidence between switching hand 0 and the switching hand with greatest total winnings. Unfortunately this improvement is due less to exploration and more to the choice of the exploration strategy. Observe that against both of these opponents, the BalancedExplore exploration strategy does better than the equilibrium if used all the way through the match. Over 200 hands, the BalancedExplore strategy has an expected value of -3.89 against O_4 and 12.5 against O_6, whereas the equilibrium has an expected value of -11.11. This suggests that using exploration strategies which can also exploit an opponent's weaknesses enables a modelling agent to improve its model with little loss due to exploration, thus yielding an overall improvement in performance. Conversely, a bad choice of exploration strategy, despite improving the model, incurs more loss than can be regained with the resulting model.

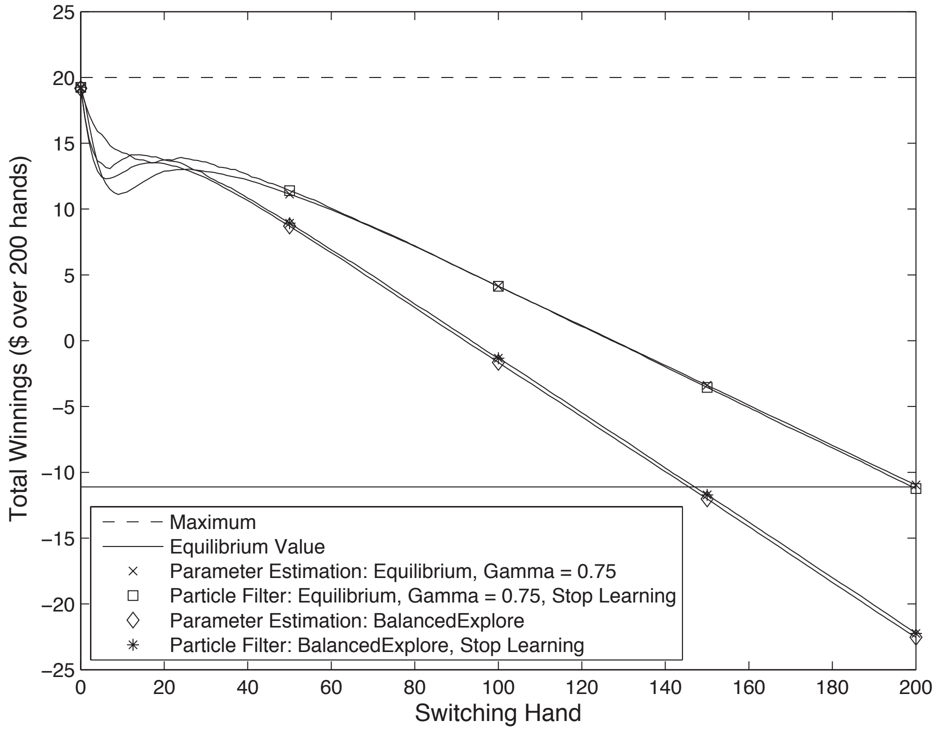
These results show that our particle filtering technique is comparable with the parameter estimation techniques presented by Hoehn and colleagues against static opponents. Moreover, it appears that continued learning not only improves performance compared to stopping learning but it also greatly reduces the benefits of an exploration phase. In fact, unless we choose an exploration strategy that also exploits the opponent, it appears that an exploration phase has little, if any, benefit over continued learning. For these reasons, we use continued learning with no exploration phase throughout the match for the rest of our experiments. Since our approach was specifically designed to handle dynamic agents we will now move on to those results.

5.2 Oblivious Dynamic Opponents

The majority of our remaining results examine the performance of our particle filtering techniques for agent modelling against a variety of oblivious dynamic opponents. Recall that oblivious opponents ignore our past actions when choosing their strategy. The oblivious opponents in our results are either switching opponents or drifting opponents using the switching or drifting models described in Section 4.2.4. We use nine distinct motion models (4 switching, 4 drifting, and 1 stationary) with the specific values of ρ and σ given in Table 5.1 to create a range of opponents that move both slowly and quickly. As with the static opponents, the particle filter's resampling phase is disabled when using the stationary motion model (model A). This choice and related discussion is presented later in the particle impoverishment segment of Section 5.2.1. In our experiments, we play several particle

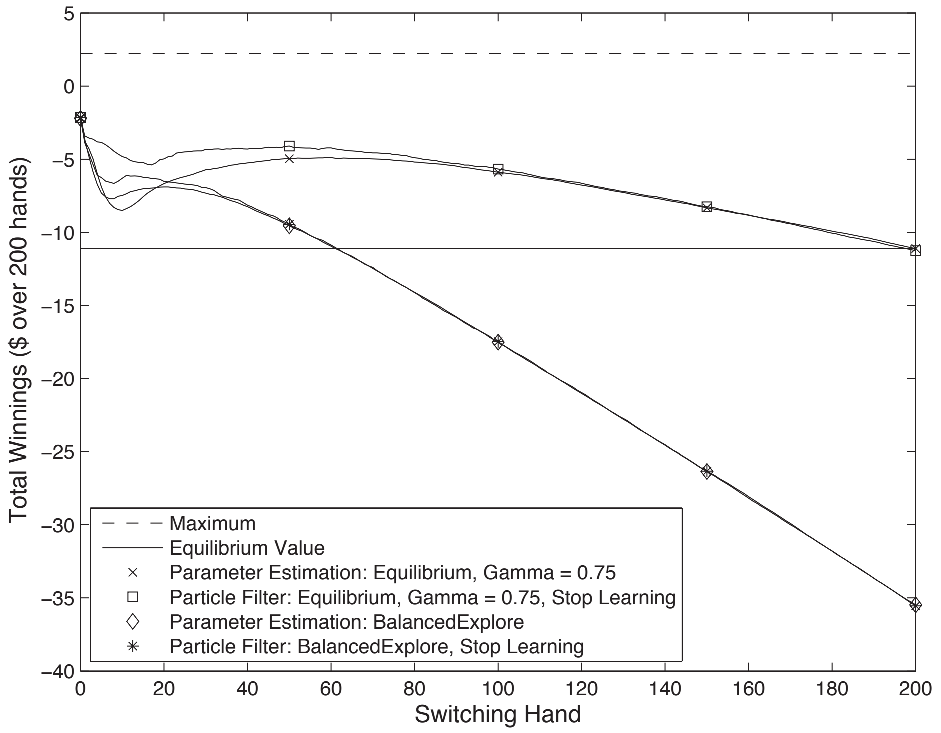


(a) $O_1 = (\frac{4}{5}, \frac{2}{7})$

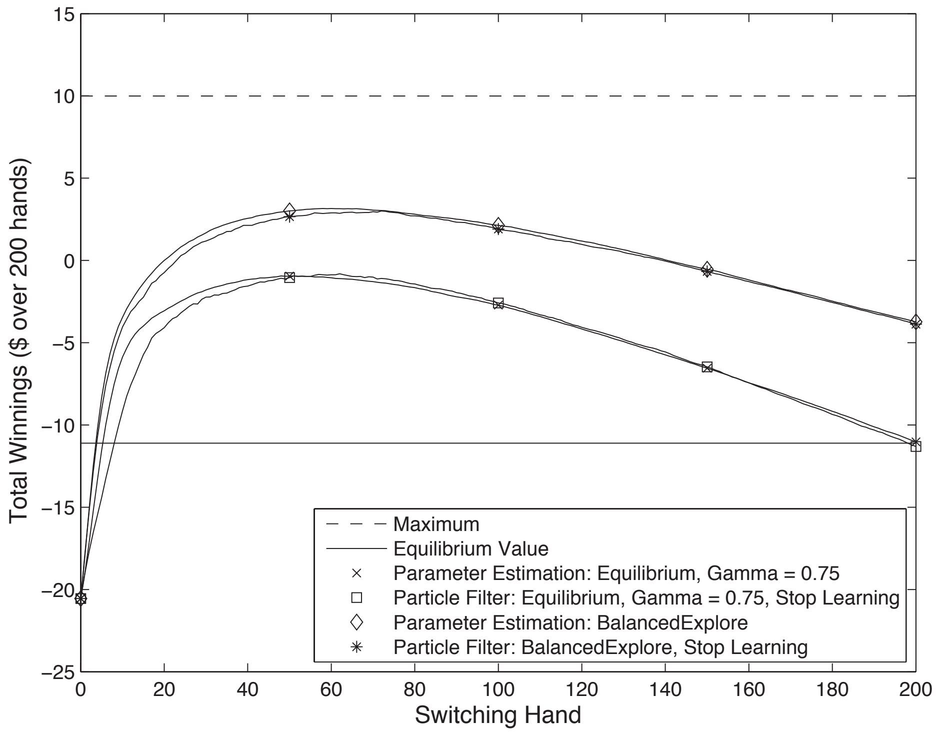


(b) $O_2 = (\frac{3}{4}, \frac{4}{5})$

Figure 5.1: Total expected winnings of Hoehn's parameter estimation and particle filtering using exploration and a stationary motion model against static opponents O_1 (a) and O_2 (b) over 200 hand matches.

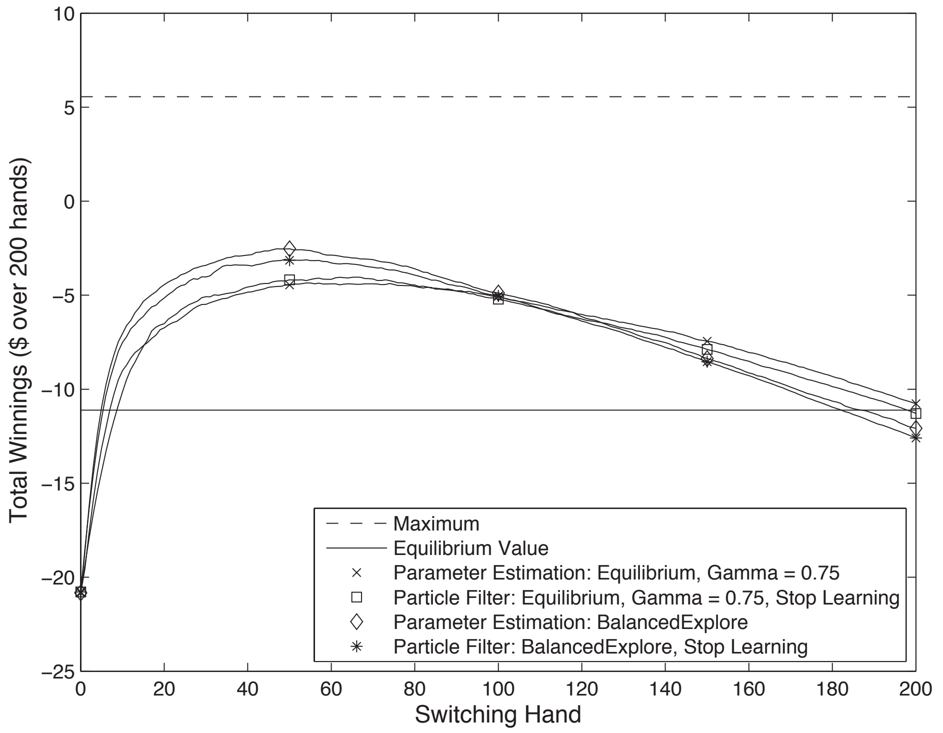


(a) $O_3 = (\frac{2}{3}, \frac{2}{5})$

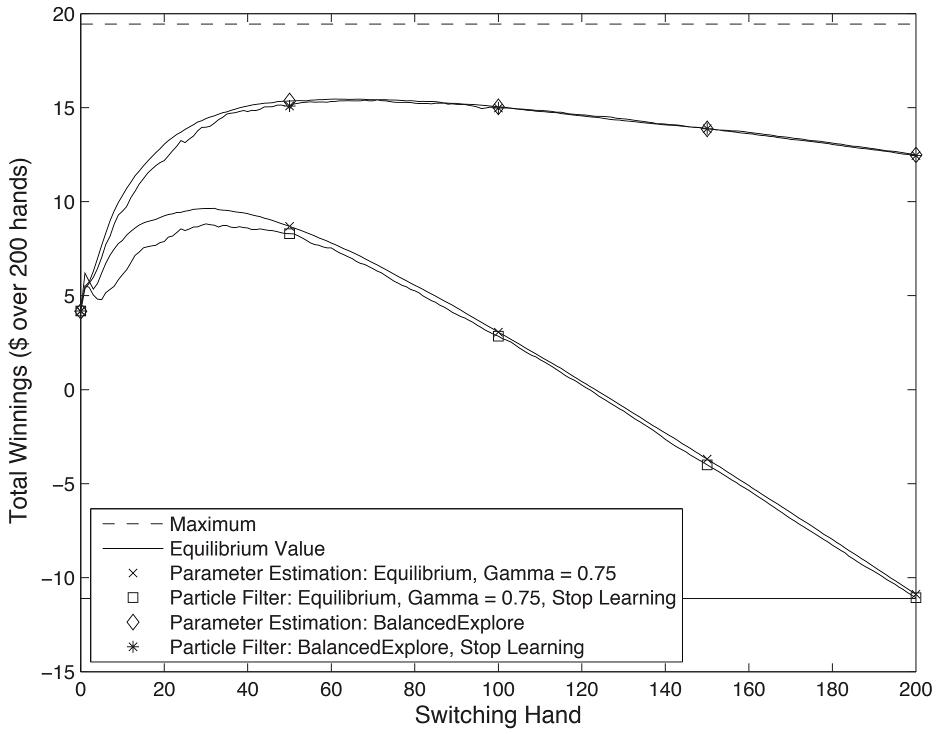


(b) $O_4 = (\frac{1}{6}, \frac{1}{5})$

Figure 5.2: Total expected winnings of Hoehn's parameter estimation and particle filtering using exploration and a stationary motion model against static opponent O_3 (a) and O_4 (b) over 200 hand matches.

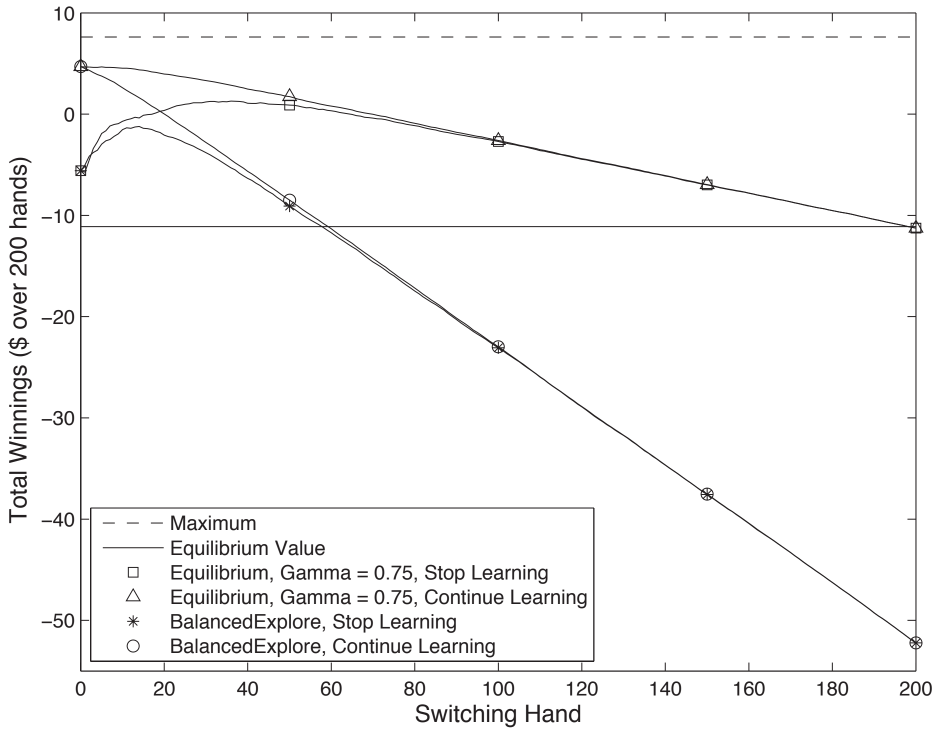


(a) $O_5 = (\frac{1}{4}, \frac{1}{6})$

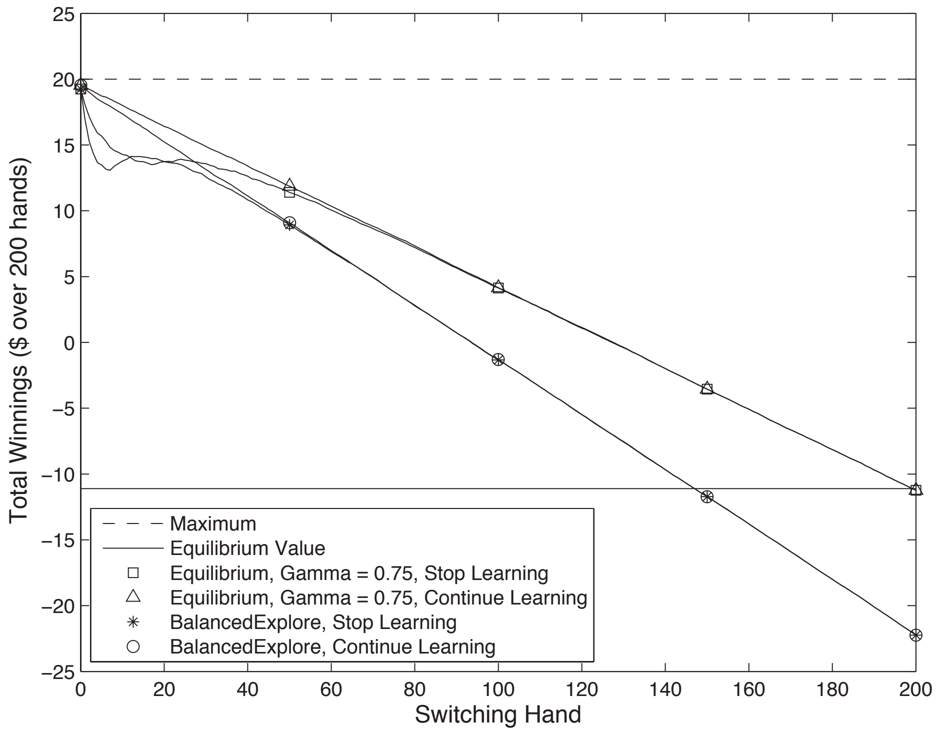


(b) $O_6 = (\frac{1}{4}, \frac{2}{3})$

Figure 5.3: Total expected winnings of Hoehn's parameter estimation and particle filtering using exploration and a stationary motion model against static opponent O_5 (a) and O_6 (b) over 200 hand matches.

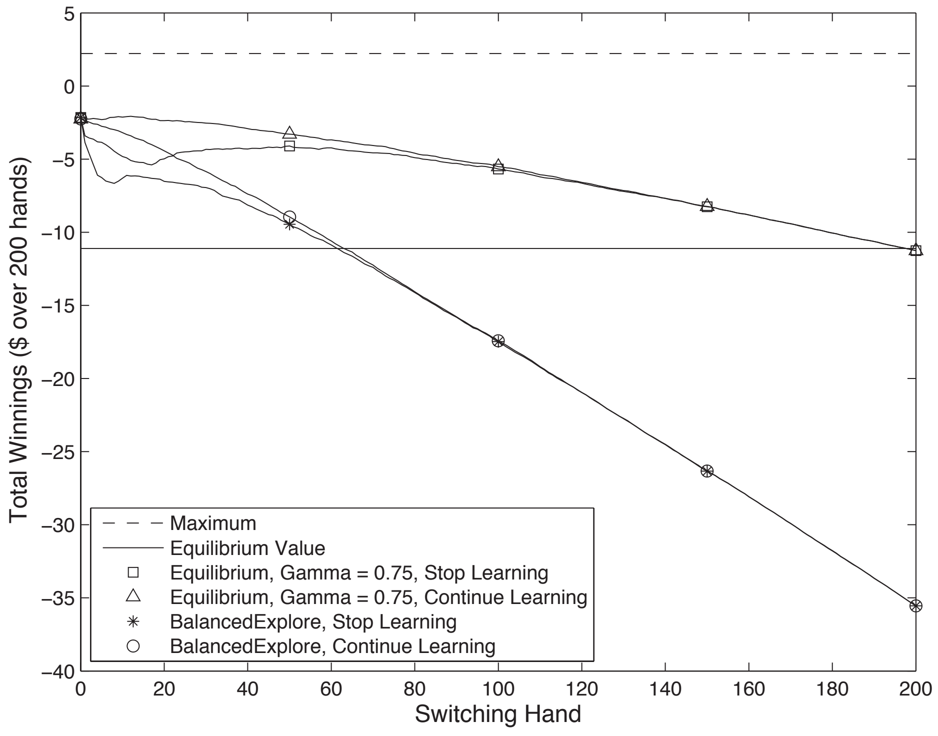


(a) $O_1 = (\frac{4}{5}, \frac{2}{7})$

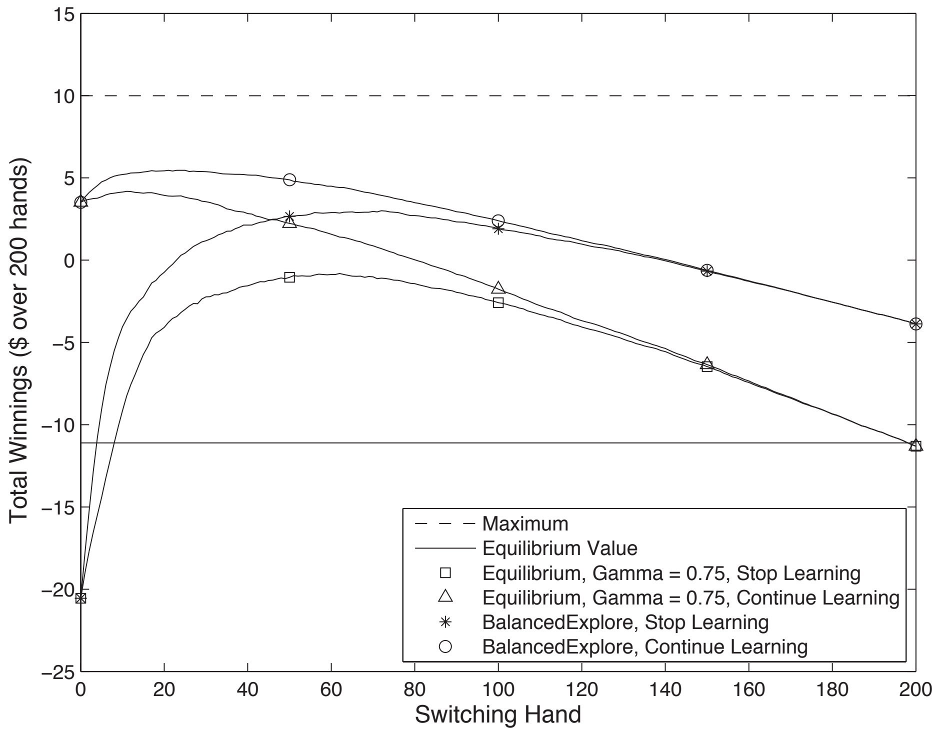


(b) $O_2 = (\frac{3}{4}, \frac{4}{5})$

Figure 5.4: Total expected winnings of particle filtering using exploration and a stationary motion model against static opponents O_1 (a) and O_2 (b) over 200 hand matches.

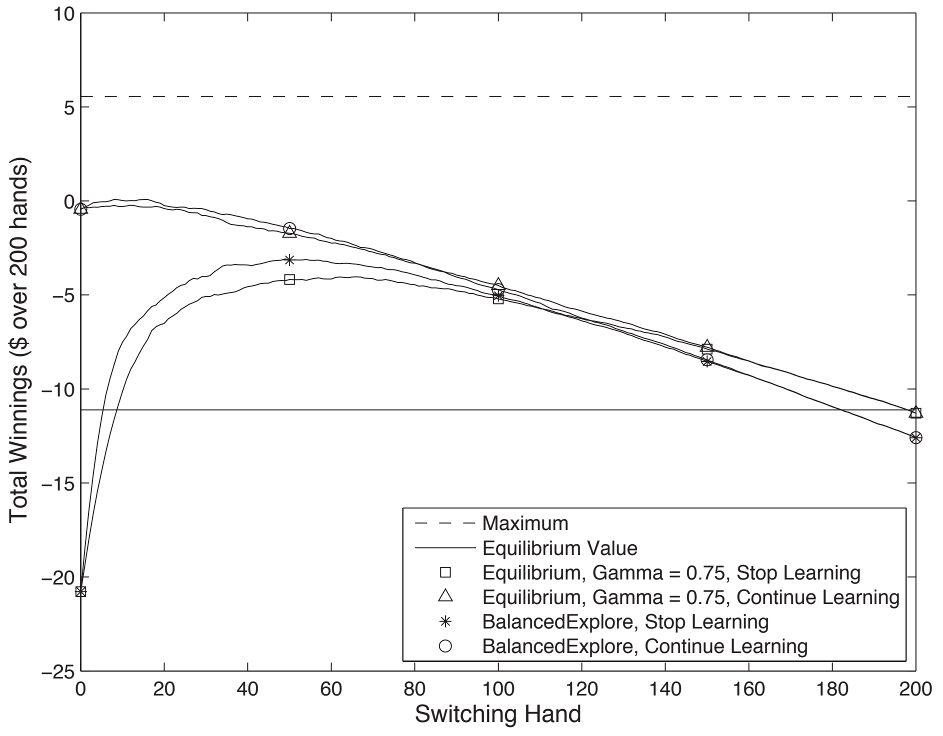


(a) $O_3 = (\frac{2}{3}, \frac{2}{5})$

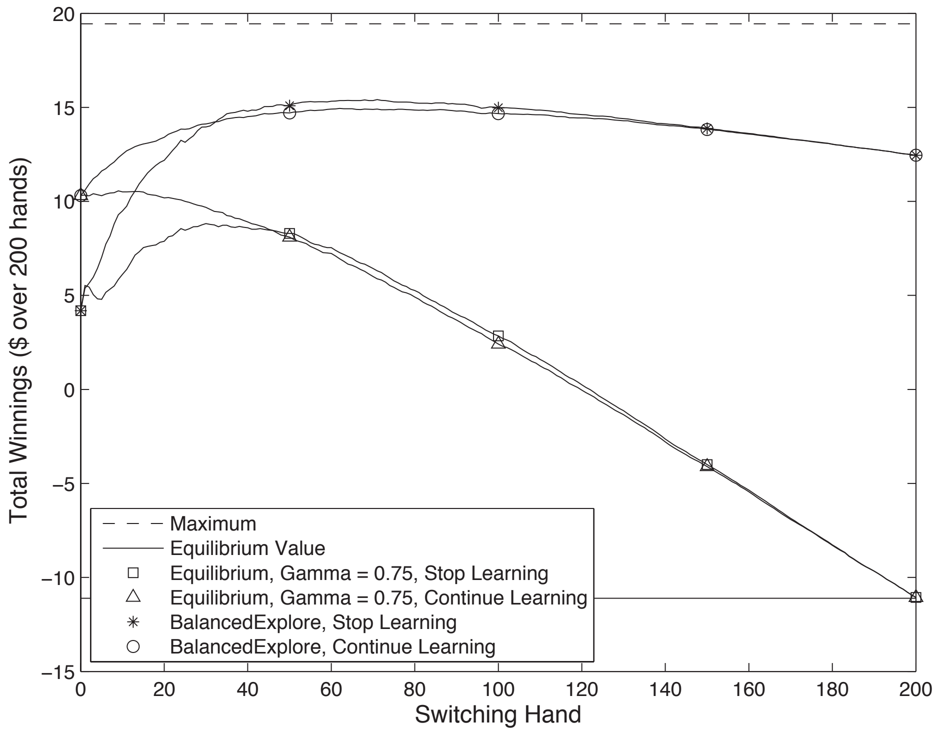


(b) $O_4 = (\frac{1}{6}, \frac{1}{5})$

Figure 5.5: Total expected winnings of particle filtering using exploration and a stationary motion model against static opponents O_3 (a) and O_4 (b) over 200 hand matches.



(a) $O_5 = (\frac{1}{4}, \frac{1}{6})$



(b) $O_6 = (\frac{1}{4}, \frac{2}{3})$

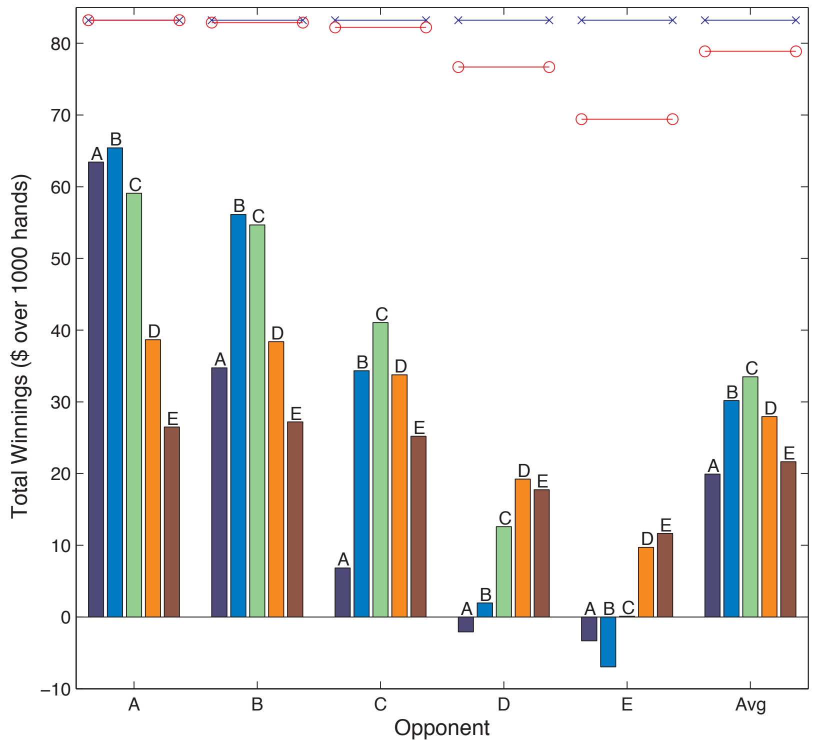
Figure 5.6: Total expected winnings of particle filtering using exploration and a stationary motion model against static opponents O_5 (a) and O_6 (b) over 200 hand matches.

filtering incarnations against each of the nine opponents for 1000 hand matches. Each particle filter uses 1000 particles with learning throughout the entire match and no exploration phase. Matches are repeated 5000 times for statistical confidence. At the beginning of each match, the opponent's η and ξ parameters are initialized by sampling each parameter independently from a uniform distribution over $[0, 1]$.

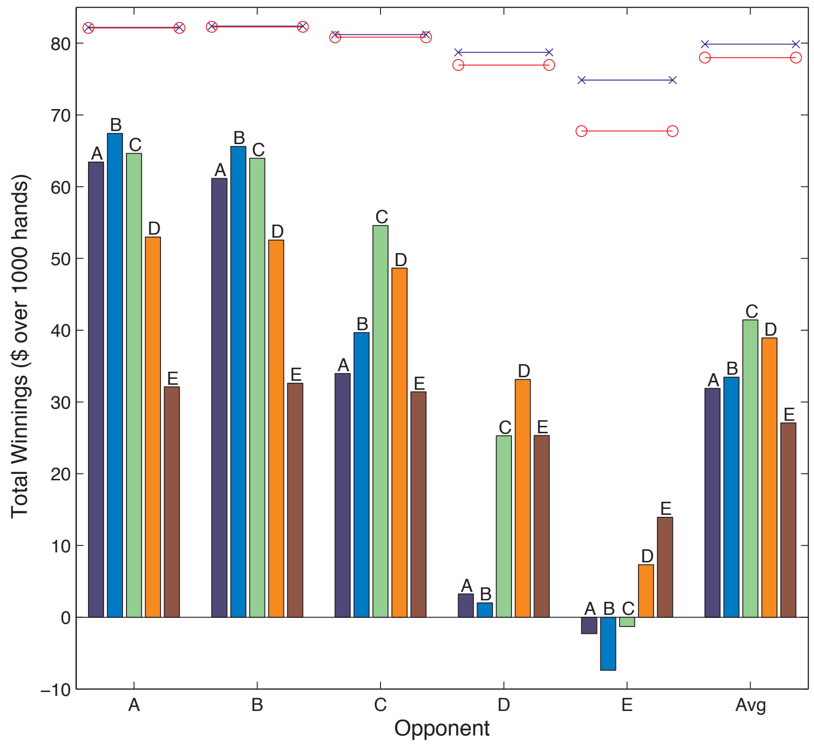
Recall that since these experiments involve player one modelling player two, this is actually a "losing" situation for the particle filter modeller. If the modelling player instead played any equilibrium strategy it would expect to lose more than \$55 over a 1000 hand match. A less naive strategy would exploit the fact that $\mathrm{E}[\eta_t, \xi_t] = (0.5, 0.5)$ with any of our motion models. Because of our choice of motion models, we can expect opponents, barring any additional information, to make uniformly random decisions between any undominated choices on every round. The best-response to such random play is an alternative static strategy that has an expected value of zero against any of our opponents. In fact, for our motion models, if the value of ρ or σ is sufficiently large, we cannot expect any agent modelling technique to do better than zero on average. This is because $\mathrm{E}[\eta_t, \xi_t | z_{1:t}]$ will approach $(0.5, 0.5)$ as ρ or σ increases. This makes zero a good baseline comparison for which effective opponent modelling should rise above.

Further context for our results is provided by two *exploitability lines*. These lines indicate the expected winnings that could be obtained given certain information about our opponent. The top line in each graph corresponds to the expected winnings if we know our opponent's strategy at every hand. This is an upper bound on any modelling agent's expected performance since it represents full knowledge of our opponent (except for their randomly dealt card). Since it is unreasonable to expect that we can attain that value, we also present a second exploitability line which accounts for uncertainty due to an opponent's strategy dynamics. The lower exploitability line represents the expected winnings if we are given our opponent's strategy on their previous hand and their motion model. Although this is a better baseline for comparison, both of these exploitability lines are loose upper bounds on the maximum possible performance since it takes time to estimate an opponent's strategy parameters even when they are static.

Our analysis examines the performance of particle filtering in three cases: when the opponent's motion model is fully known, when the model's form is known but the parameters are not, and when the exact model form is not known. Figures 5.7 and 5.8 summarize the results of using the various incarnations of particle filters against the nine aforementioned opponents. Graph (a) of both figures shows the results for the switching opponents and graph (b) shows the results for the drifting opponents. In all of the graphs, the x-axis corresponds to the different opponents from Table 5.1. Note that opponent A is stationary and therefore the same in the switching opponent and drifting opponent graphs. The set of bars for each opponent shows the average total winnings of one of the particle filter variants. The "Avg" opponent is the average result against opponents A through E. We dissect these figures further throughout the rest of this section.



(a) Switching Opponents



(b) Drifting Opponents

Figure 5.7: Total winnings of vanilla particle filter modelling against different oblivious dynamic switching opponents (a) and drifting opponents (b). Each bar represents a different setting of ρ (a) or σ (b) in the particle filter's motion model

5.2.1 Known Motion Model

For each opponent in Figure 5.7, the set of bars shows the average total winnings of particle filtering when using the correct form of the motion model, but varying the parameter in that model (*i.e.* ρ or σ). We use the same set of values for the particle filter parameters as the opponents and so each bar is labeled with the corresponding opponent's identifier from Table 5.1.

Consider the case when we know both the form and the parameters of the opponent's motion model. Specifically, we know whether the opponent is a switching or drifting player as well as the exact probability of switching or the standard deviation of their drift. This corresponds to the bars of Figure 5.7 that match the opponent being played on the x-axis. Although the winnings vary depending on the opponent, for all nine opponents the correct model outperformed the baseline static strategy which has an expected value of zero.

In reality, it is impractical for us to assume that we would ever have a completely correct model of our opponent. This leads us to wonder what would happen if our model was wrong. The remaining bars of Figure 5.7 correspond to using the correct form of our opponent's motion model but with an incorrect parameter value. Generally, our results make intuitive sense. The closer the particle filter's parameter is to the true parameter value, the better it performs. Conversely, greater differences between the parameter and the true value result in poorer performance. Sometimes an incorrect model can cause a drastic decrease in performance, even dropping below the static baseline's expected value of zero. For instance, against both the switching and drifting opponents we see that if we believed the opponent to be using model B and they were actually using model E, then we drop below our baseline. Despite this poor performance, it is interesting to note that this worst case scenario still outperforms an equilibrium strategy by over \$48. There are a few exceptions to these general trends which are caused by *particle impoverishment*. This phenomenon is discussed next.

Particle Impoverishment

At a high level, *particle impoverishment* occurs when the particle filter becomes overconfident in its posterior distribution. A particle filter is *overconfident* if most of the particles' weight is concentrated in only a few places (possibly just one). The primary cause of particle impoverishment is insufficient noise in either the motion or observation model to maintain a diversity of particles. Because of this, even an accurate probabilistic model of a system with little noise can still lead to overconfidence and particle impoverishment.

Consider the case when we make an observation where our observation model assigns a large weight to particles in a small region of the state space and a small weight to the rest of the particles. If we resample after such an observation, then some low weight particles will be discarded in exchange for particles with large weights. This is fine so long as there are enough distinct particles to approximate the true posterior distribution. On the other hand, if many of the particles are collocated then we may not have enough remaining particles to accurately approximate the low, but non-zero, probability regions of the posterior.

Using a motion model that adds noise to the particles, even if the noise is artificial, will help to keep the particles distinct and prevent particle impoverishment. A stationary motion model on the other hand will inevitably lead to particle impoverishment. If the particles never move except during resampling, then we will progressively remove states that are covered by a particle. We attempted to mitigate this problem in our experiments by disabling the resampling phase when using the stationary model.

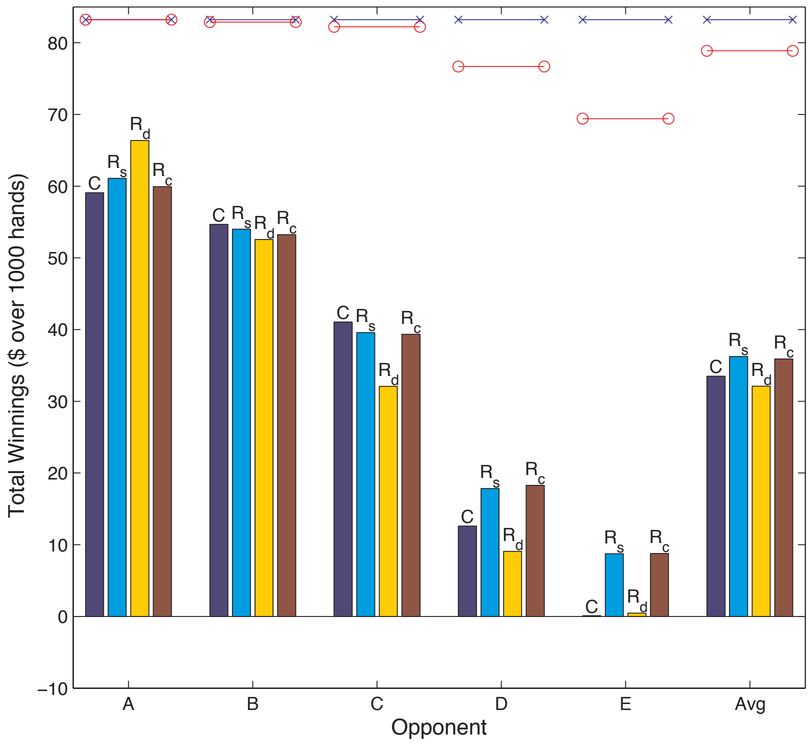
Particle filters can still become overconfident for other reasons. In our experiments, using the stationary motion model with no resampling results in the particle filter creating fixed particles whose weights are updated throughout the match. If one particle is considerably more probable than the others, it will take many observations to overcome the weight built up by the initial observations. We believe this is why model A does poorly when modelling non-stationary opponents in Figure 5.7. Moreover, when the particles are stationary they cannot be moved later to refine our estimate. This means that we are relying on starting with a particle close to our opponent's true strategy. If this does not happen then our agent's performance will suffer. This is probably the case in our experiment comparing model A against opponent A in Figure 5.7. Using "noisier" models can improve a particle filter's robustness to overconfidence. In our case, motion models with larger values of ρ and σ are more likely to create new particles throughout a match. This gives the particle filter a chance to recover when it is wrong. The effects of this improved robustness is observed in Figure 5.7 where the "noisier" motion models tend to outperform model A despite being further from the opponent's true motion model parameters. Similar phenomenon have been well documented in the particle filtering literature [17].

Figure 5.7 also shows us that noisier models are not always better. Note that the stationary model A outperforms model B against some of the quickly moving opponents (especially E). Although noisier models tend to be more robust, model B adds very little noise – insufficient noise to prevent particle impoverishment. Since model A does not resample, and the opponents move quickly, model A does not become as overconfident as it would against slowly changing opponents. This results in model A being a more robust choice than model B against these opponents.

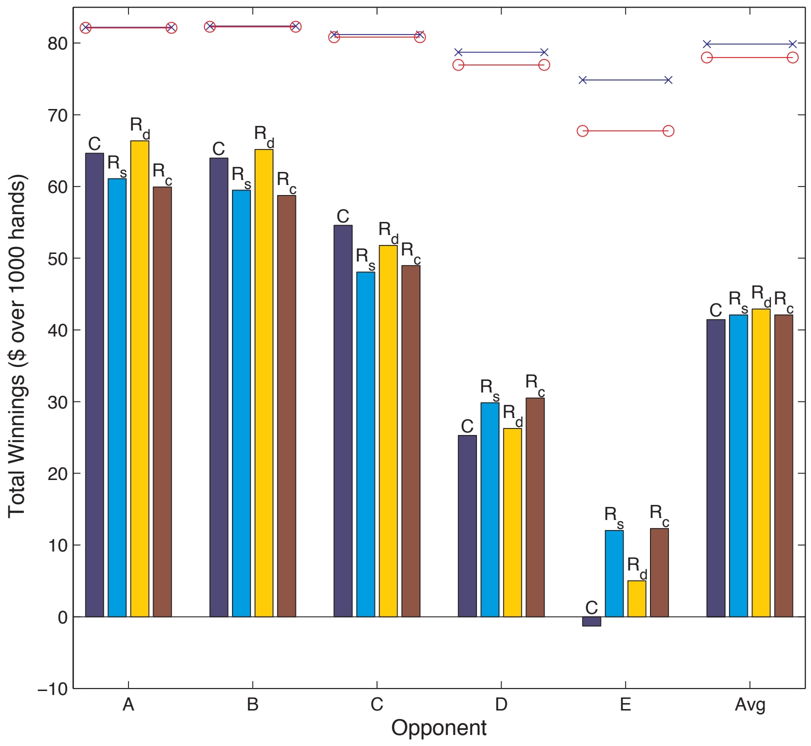
5.2.2 Unknown Motion Model Parameters

Since it is unrealistic to expect to know the exact parameters of an agent's motion model, we need a more practical way to model that is also robust to error. One simple approach is to use a standard particle filter with a motion model parameter that works well even when incorrect. Figure 5.7 indicates that model C has the best average performance across both the switching and drifting opponents (see the "Avg" bars at the right of the graphs).

Alternatively, we could use the Rao-Blackwellized particle filtering technique described in Section 3.2.3 for dual estimation of the state and the motion model parameter. The results of this



(a) Switching Opponents



(b) Drifting Opponents

Figure 5.8: Total winnings of RBPF modelling against different oblivious dynamic switching opponents (a) and drifting opponents (b). The bars labelled “C” come from Figure 5.7.

approach are shown for the same suite of switching and drifting opponents as before in Figure 5.8. For each opponent we present model C from Figure 5.7 as a baseline for comparison along with three RBPF modellers. R_s corresponds to using our prior over switching probabilities (assuming no drift), R_d corresponds to using our prior over drifting amounts (assuming no switching), and R_c corresponds to an independent prior over both forming a combined model. For the moment our analysis will focus on R_s and R_d, leaving discussing about the combined model until the next section.

Examining the graphs of Figure 5.8 we observe that against most switching opponents R_s performs better than the baseline of particle filtering using model C. Although model C does outperform R_s occasionally, R_s is never dramatically worse than model C. Against the quickly switching model E opponent, R_s does considerably better than model C since the RBPF technique enables R_s to infer the opponent's switching parameter. This means that R_s is able to learn and adapt to the fast switching rate of opponent E whereas model C is locked into its one parameter value. Although the RBPF techniques only yield marginal performance improvements on average, they do provide additional flexibility and robustness. More specifically, using the RBPF technique means we do not need to rely on "guessing" the best fixed model for arbitrary agents. Comparing model C and R_d against the drifting opponents yields a similar result.

Once again, it is interesting to consider the effects of having incorrect information when modelling since it is unlikely that we can rely on agents being pigeonholed into a small number of behaviours. Going one step further than in Section 5.2.1, we examine the impact of having the incorrect form of our opponent's motion model. This type of error is observed in Figure 5.8 when using R_d to model switching opponents and R_s to model drifting opponents.

Despite having the incorrect form of the model the loss incurred by this error (relative to using the correct RBPF model) is never more than \$9 over 1000 hands. This is relatively small considering that none of our beliefs about the dynamics are correct. Also note that the average performance of our dual estimation with the wrong form of the model is still close to the average performance for model C – our best average model when particle filtering with the correct model form from Figure 5.7. This result is due to the RBPF learning a parameter for the incorrect model form that describes the opponent's dynamics relatively well. In fact, we can see that R_s actually does better than R_d at modelling quickly drifting opponents. In this specific case, our choice of prior probably makes the switching model assign a higher likelihood than the drift model to large strategy drifts. Our effective modelling, despite incorrect models of our opponents, suggests that Rao-Blackwellized particle filtering is robust enough to compensate for both an incorrect model parameter and an incorrect form of the model.

5.2.3 Unknown Motion Model

In general, it is impractical to assume that we have correct information about an agent's dynamics model. Regardless of this problem, we want to model agents effectively while remaining robust to potential errors. If we do not know the exact form of the model we can still employ RBPF techniques for dual estimation. In the same way we used RBPFs in Section 5.2.2 to infer the single parameter of a given motion model, we can also use RBPFs to infer multiple parameters of a combination of model forms. In this experiment we use RBPFs to infer the parameters of our combined motion model as described in Section 4.2.4.

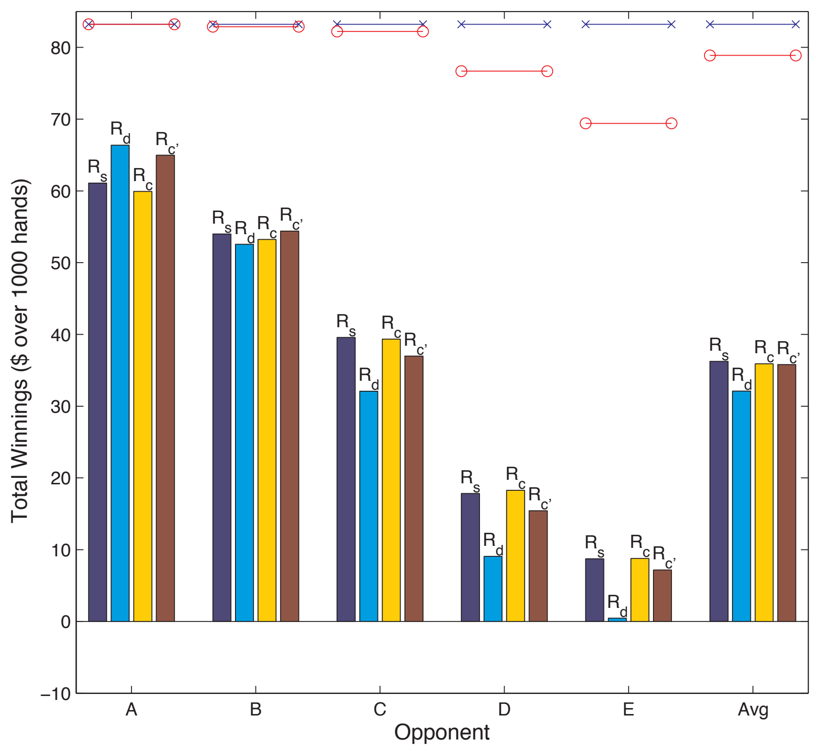
Turning now to Figure 5.8, we see that the combined model, R_c, is reasonably competitive with the correct model against all opponents. One slight oddity in the results is that R_c performs almost identically to R_s. We believe that this similarity is due to our choice of prior over the two models. This claim is supported by Figure 5.9 which shows the performance of R_s, R_d, R_c, and another combined model $R_{c'}$. The prior for $R_{c'}$ is identical to the prior for R_c mentioned in Section 4.3.2 except that $R_{c'}$ uses a "lighter" prior with $\rho \sim \textsc{Beta}(0.1, 3.0)$. Both of these priors have the same mean, but the prior for $R_{c'}$ has a larger variance and is more easily overcome by new observations than the prior for R_c. Generally R_c performs better against opponents that change strategies more quickly. This intuitively makes sense since having stronger beliefs that our opponent is moving rapidly should improve the modelling against opponents who are, in fact, moving rapidly.

These results suggest that RBPFs can also account for an unknown form of an agent's motion model. This is satisfying since it means that it is not critical to know much about our opponent a priori – just a few models that could explain their dynamics. One question that these results do not answer is how well our approach works against opponents who are not oblivious and will pay attention to our actions when making their decisions. The next section addresses this question.

5.3 Non-oblivious Dynamic Opponents

Up to this point all of our results have focussed on opponents that do not learn and either use a fixed strategy or randomly alter their strategy. These opponents provide a good baseline evaluation, but they fail to test the performance of our technique against more sophisticated opponents. We would especially like to fare well when modelling adaptive agents since it is doubtful that a human player will be oblivious. For our experiments against non-oblivious opponents we compare our technique against an agent using the Exp3 regret minimization algorithm presented by Auer and colleagues [1] (and also examined in Hoehn's work [12]).

As we mentioned in Section 2.5.1, the Exp3 algorithm is an experts algorithm for regret minimization. On each trial, an experts algorithm updates the probability distribution over its experts based on the amount of reward each expert received (or would have received) on the iteration. More specific to our application, Exp3 updates its distribution over experts after receiving the outcome (money won or lost) of each hand. The Exp3 agent will play as player 2 modelling player 1 (our agent modeller). In this case, the Exp3 agent's experts will consist of the four best response strategies available to player 2 (*i.e.* player two's pure strategies). Our implementation of the algorithm effectively mirrors the description found in [1].


(a) Switching Opponents

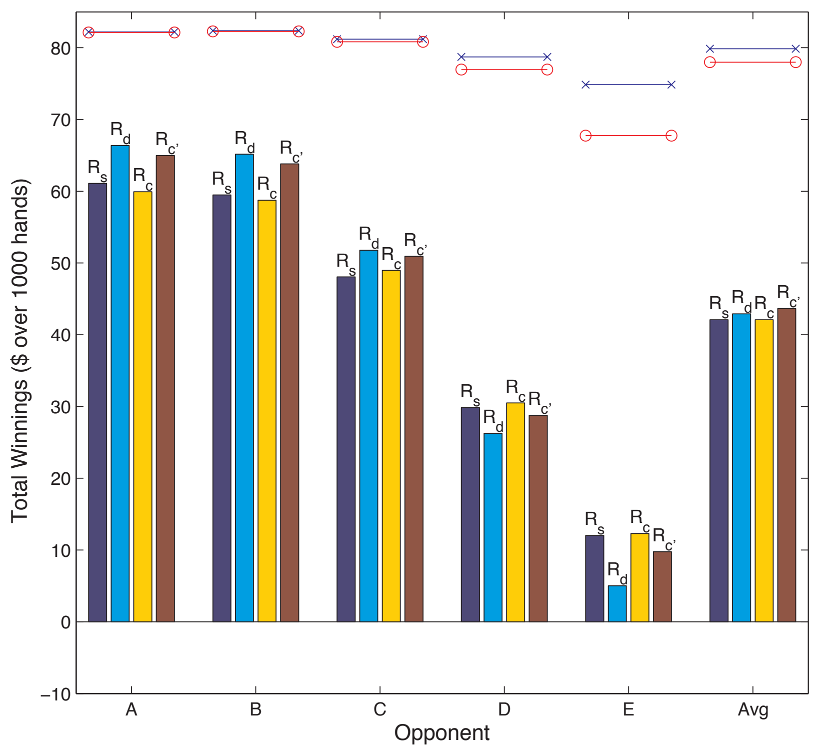

(b) Drifting Opponents

Figure 5.9: Total winnings of RBPF modelling against different oblivious dynamic switching opponents (a) and drifting opponents (b). The bars labelled R_s, R_d, and R_c all come from Figure 5.8. $R_{c'}$ was generated using a lighter $\rho \sim \textsc{Beta}(0.1, 3.0)$ prior.

One detail that was left to us was a choice of parameters for the algorithm. The Exp3 algorithm presented in [1] required two parameters: a parameter to control the amount of uniform exploration, γ, and a parameter to control the learning rate, η. These parameters are not to be confused with the Kuhn poker strategy parameters. To set the parameters, we used the method suggested in [1] that would preserve the regret bounds. For our case of using the Exp3 algorithm as player two modelling our algorithms as player 1, this worked out to a γ of 0.056808 and η of 0.014202.

Exp3 is the simplest experts algorithm for a game with partial information. Unlike more sophisticated experts algorithms, Exp3 only rewards the expert that was actually being used in a given hand. In contrast, other experts algorithms (*e.g.* Exp4) also update experts that are related to the chosen expert. This is done by checking which experts would have made similar decisions to the chosen expert. For the purposes of this thesis, we chose to use Exp3 as our non-oblivious opponent due to the ease of implementation.

Our results display two baselines to provide context for our results. The first baseline is the equilibrium value of the game (-55.56 dollars over 1000 hands). This is displayed as the minimum value in the graphs of Figure 5.10. Note that this is different from our previous static strategy baseline with an expected value of zero. Since Exp3 is not oblivious, we cannot exploit the fact that our opponent's strategy distribution will have a fixed expected value. In fact, Exp3's *expected regret* (the difference between the total reward of Exp3's best expert and the expected reward of Exp3) is bounded by $O(\sqrt{T})$ where T is the number of trials (*i.e.* hands of Kuhn). Therefore, as T approaches infinity, Exp3's average expected regret $\sqrt{T}/T$ will converge to 0 and Exp3's strategy will win at least as much as the equilibrium. Our second baseline for these experiments is another exploitability line. In this case, the exploitability line indicates the expected outcome of the game if we are given the weighted mixture of Exp3's experts on every hand. This exploitability line amounts to knowing the internal state of Exp3 (each of its experts and their associated probability of being chosen) but not the random number generator used by Exp3. The anomalous value for the exploitability against drifting model D has been confirmed with repeated experiments. Unfortunately, at this time we have no explanation for why the exploitability of Exp3 increases for this model while the particle filter performs more poorly.

Figure 5.10 shows the performance of our different particle filters and RBPFs against the Exp3 algorithm. These experiments share the same experimental setup as those in Section 5.2: 1000 hand matches repeated 5000 times where each filter uses 1000 particles with learning throughout the entire match and no exploration phase.

Observe that in contrast to the oblivious opponents, all of our techniques (regardless of motion model) lose money against the Exp3 player. Keep in mind that Kuhn poker is a "losing" game for player one with an equilibrium value of -55.56 over 1000 hands. Moreover, even if we were given Exp3's weighted mixture of experts, we still would have lost between -4.55 and -7.9 over the match. That said, all of our techniques still outperform an equilibrium strategy by a large margin –

reiterating why agent modelling is beneficial against exploitable opponents. Also note that none of the models we used were specifically crafted to model our Exp3 opponent. This suggests that simple motion models can still be effective at modelling more complex dynamics.

Against both oblivious and non-oblivious opponents, regardless of having the correct form of the model or the correct parameters, state estimation techniques provide an effective and robust framework for modelling dynamic agents in Kuhn poker. One final consideration with any particle filtering algorithm is how many particles are needed to perform well. This topic is the focus of the next section.

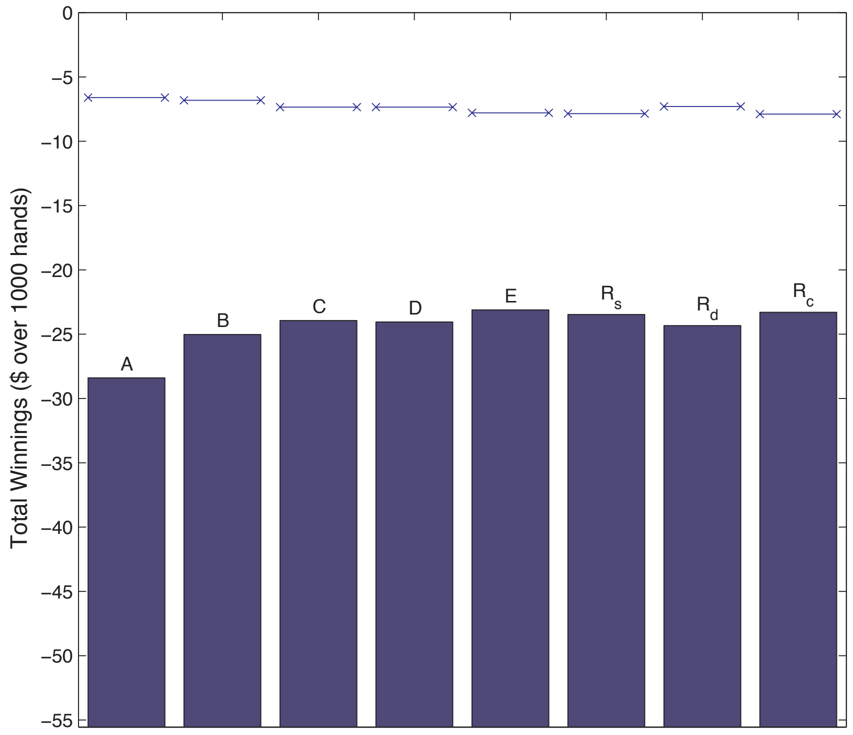
5.4 Effect of the Number of Particles

The quality of a particle filter's approximation depends, in part, on the number of particles used to represent the posterior. Examining the quality of our particle filtering algorithms when using different quantities of particles may provide answers to two important questions. What is the smallest number of particles we can use while maintaining good modelling? If we could use more particles, how much would it improve our modelling?

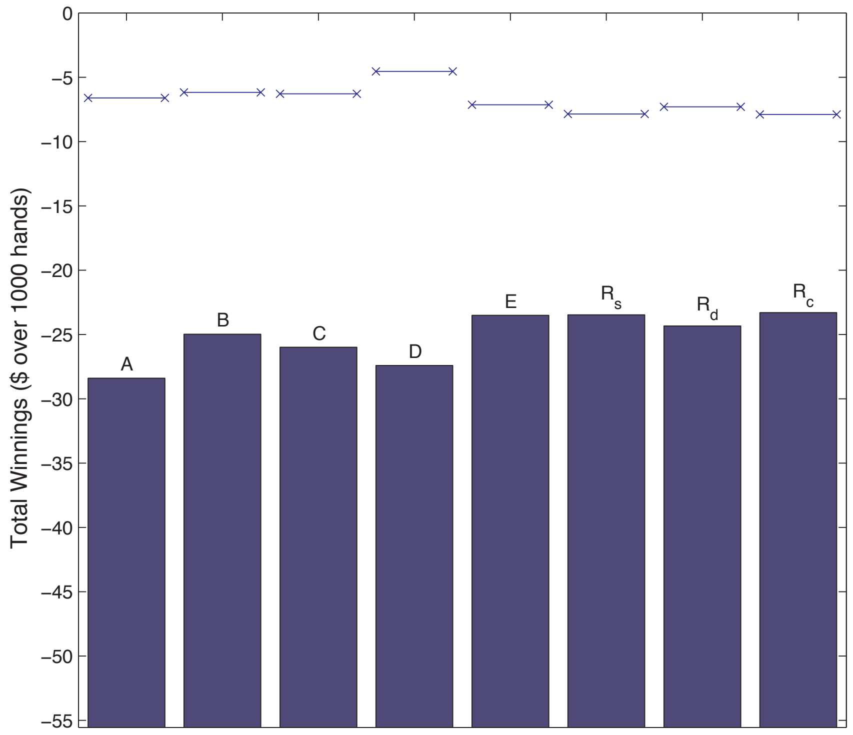
These experiments compare the performance of model C – our best average particle filter model – against oblivious opponents using our nine different models (4 switching, 4 drifting, and 1 stationary). Experiments consisted of 5000 trials of 1000 hands without exploration and learning continuing throughout the entire match. As before, the opponent's η and ξ parameters are initialized according to a uniform distribution over $[0, 1]$. Results with up to 500 particles are shown in Figure 5.11 and up to 10000 particles is shown in Figure 5.12. We first consider the expected effect of varying the number of particles and then contrast this with our experimental results.

Typically, one expects that increasing the number of particles in a particle filter will improve the performance of the filter, although with diminishing returns. This intuition stems from the fact that particle filters are known to be asymptotically unbiased [25]. That is, as the number of particles approaches infinity, the particles will be distributed according to the posterior. On the other hand, if the number of particles is finite, then the sampling process will result in a biased estimate of the true posterior. This bias is especially evident when using a small number of particles. To illustrate this, consider the extreme case where there is only a single particle. This particle will be sampled from the motion model, weighted by the observation model, and resampled. Since there is only one particle, the resampling phase will always result in the single particle being chosen, regardless of its weight. This effectively discards the observation model weighting. As a result, the particle (and therefore our estimated posterior) will be distributed according to $\Pr(x_t|u_{1:t})$ instead of $\Pr(x_t|z_{1:t}, u_{1:t})$. This type sampling bias can frequently have a negative impact on the particle filter's estimate.

The intuition that increasing the number of particles will yield diminishing returns holds true experimentally. Figure 5.11 shows a trend of large performance gains from relatively small increases in particles. Diminishing returns are evident in Figure 5.11 – especially when the number of particles



(a) Switching Models



(b) Drifting Models

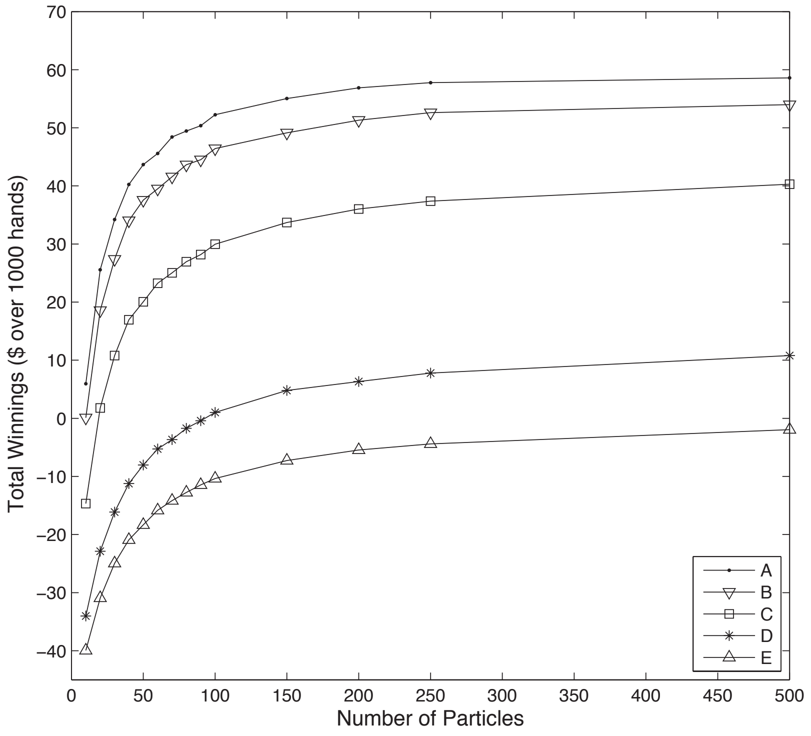
Figure 5.10: Total winnings of RBPF modelling and vanilla particle filter modelling with switching models (a) and drifting models (b) against the non-oblivious dynamic Exp3 opponent.

is increased from 250 to 500. Looking at Figure 5.12, we see that diminishing returns continue up to the point of 10000 particles.

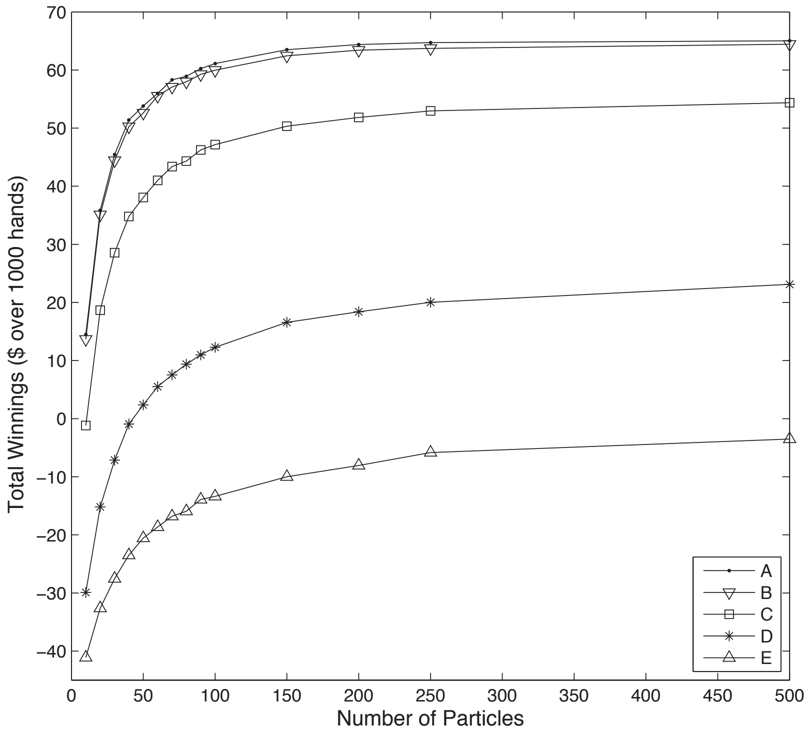
One counterintuitive result is the decrease in performance when increasing the number of particles used against opponents A and B (though the decrease is very slight against switching opponent B). As we demonstrate in the following example, the change in bias resulting from an increase in particles does not necessarily move our estimated mean closer to the system's true state. Suppose that we have a set of particles modelling a state whose value is in $[0, 1]$, like our Kuhn parameters η and ξ. Assume that we saw an observation that corresponds to the true state being $x = 1$ and the particle filter's observation model returns a likelihood equal to the particle's state (*i.e.* particle p with state $x \in [0, 1]$ has unnormalized probability x). Normalizing this distribution so it sums to 1 over the range of x results in $\Pr(z|x) = 2x$. If we had an infinite number of particles that were uniformly distributed in $[0, 1]$, then the mean of the particles would be $\int_0^1 \Pr(z|x)x\, dx = \int_0^1 2x^2\, dx = 2/3$. Now consider the case where we have a finite set of particles, say three of them, distributed with $x = \{0, 0.5, 1\}$. Then by taking the weighted mean of the particles we get a mean of $(0^2 + 0.5^2 + 1^2)/(0 + 0.5 + 1) = 5/6$. Similarly, recomputing the mean for particles at $x = \{0, 0.25, 0.5, 0.75, 1\}$ yields a mean of $3/4$. As we can see from this example, using a small number of particles can bias our estimated mean toward the true value of 1 and away from the unbiased mean of $2/3$. Even the increase from 3 to 5 particles results in a significant shift away from the true value.

We believe that the dip in our performance after an increase in particles is caused by the same effect. Consider a strategy for player two that is close to the boundaries of the $[0, 1]^2$ strategy space. Since it is close to the boundary of the strategy space, it is likely that the distribution of particles around the strategy will be uneven. Like the above example, an uneven distribution of particles can shift the mean of the particle filter away from the true strategy. This effect is especially apparent against opponents A and B since they move slowly and are more likely to initially choose a strategy near the boundary and then never move from that strategy.

Our choice to use 1000 particles in our earlier experiments was fairly arbitrary and largely based on the time required to run experiments. From these results, we can see that increasing the number of particles yields very little improvement. Moreover, if we were concerned about the computational complexity of the modelling and were willing to suffer some performance losses, then we could use between 250 and 500 particles while still maintaining relatively good performance. In fact, for model C against the oblivious opponents, using only 10 particles still beats an equilibrium strategy.

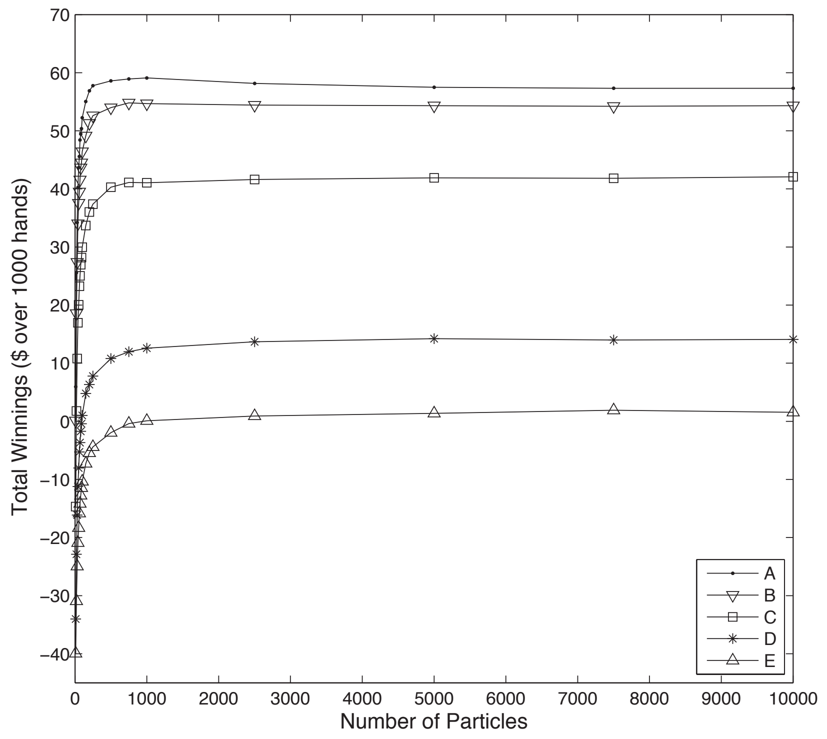


(a) Switching Opponents

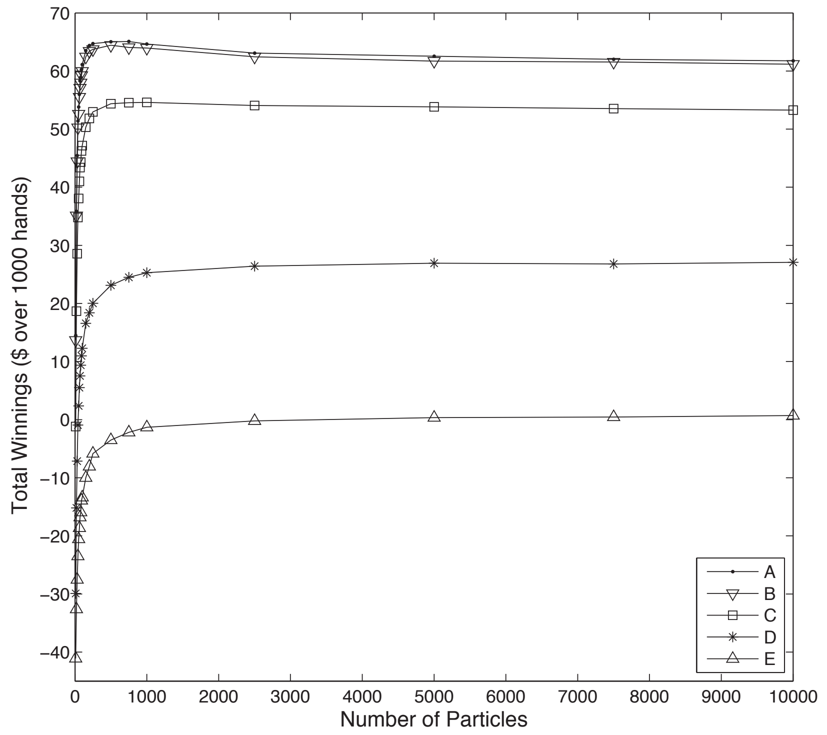


(b) Drifting Opponents

Figure 5.11: Total winnings of particle filter modelling using switching model C (a) or drifting model C (b) against different oblivious dynamic switching opponents (a) and drifting opponents (b). Each datapoint represents a different number of particles used in the particle filter.



(a) Switching Opponents



(b) Drifting Opponents

Figure 5.12: Total winnings of particle filter modelling using switching model C (a) or drifting model C (b) against different oblivious dynamic switching opponents (a) and drifting opponents (b). Each datapoint represents a different number of particles used in the particle filter.

Chapter 6

Conclusions and Future Work

This thesis advocated the use of state estimation techniques as an approach to modelling agents with dynamic behaviour. We described two well known Monte Carlo state estimation algorithms: particle filters and Rao-Blackwellized particle filters. We then described how we applied these techniques for modelling agents in the domain of Kuhn poker. Our experiments showed that these techniques were effective at modelling and exploiting static and dynamic opponents. Moreover, our experiments demonstrated that using Rao-Blackwellized particle filters for dual estimation of an opponent's strategy and dynamics provides robustness to incorrect beliefs.

Although equilibrium strategies guarantee a minimal expected payoff, they fail to fully take advantage of weaknesses in an opponent's strategy. We demonstrated in our experiments that our approach provides considerable improvement over equilibrium policies when playing against exploitable opponents. Furthermore, our experiments showed that our state estimation approaches were competitive with the approaches presented by Hoehn *et al.* [12] for modelling static opponents in Kuhn poker.

Unlike many agent modelling techniques, including Hoehn's approach, our focus was on modelling dynamic agents rather than static agents. Our state estimation approach allows us to directly encode an agent's dynamics into our model. This makes it easier to express agent dynamics that are more sophisticated than simply "forgetting" past observations. Modelling an agent's dynamics distinguishes our approach from many previous agent modelling techniques that would fail if the agent's behaviour changed over time. Our use of RBPFs for dual estimation also enables us to learn the parameters of an agent's dynamics online. This allows us to focus on understanding high level agent dynamics rather than low level parameter tuning.

This work focusses on the small domain of Kuhn poker. Kuhn poker provides an ideal imperfect information domain for testing our agent modelling approaches. Its small size, compact strategy representation, and known equilibrium value provides a domain with interesting properties but without many other difficulties that would complicate our evaluation. Although there are challenges to using state estimation algorithms for general agent modelling, we believe that many of these challenges can be overcome. Moreover, we believe that these techniques can be effectively applied to larger domains including Texas Hold'em poker. The rest of this chapter will address some of these challenges, their solutions, and directions for future work.

6.1 Challenges

Our application of state estimation techniques in Kuhn poker fails to show how to use these techniques in larger domains. Scaling up our approach to larger domains presents some challenges.

6.1.1 Computational Cost

First and foremost among the challenges is the computational cost of using our particle filter and RBPF approaches for agent modelling in "large" domains. The cost of these filters is proportional to the number of particles used. Increasing the dimension of the state space being modelled requires exponentially more particles to cover the space. This "curse of dimensionality" means that these techniques cannot be used when the dimensionality of the state space becomes "large". At this point it is not clear how large a state space can be while remaining tractable for particle filters or RBPFs. Previous work in robotic localization has shown that particle filters can easily model a 3 dimensional space. But how well would these techniques do in, say, a 6 dimensional space? How many dimensions could we model before our performance suffers? The answer to this question will depend in part on the problem domain. If domain performance is relatively insensitive to errors in the estimated state, having a very sparse set of particles may not be a problem. Conversely, if the performance was very sensitive, it could be disastrous.

One possible solution to this problem is that there are a number of other Bayesian filtering techniques which scale up more easily. Some examples are Kalman filters, extended Kalman filters, and sigma-point Kalman filters (unscented Kalman filters). These algorithms make limiting assumptions about the shape of the posterior distribution or the system's dynamics. For instance, all of the aforementioned examples assume the posterior distribution can be represented with a Gaussian distribution. This assumption limits their representational power, but reduces their computational cost so it does not grow exponentially with the dimension of the space. If the specific properties of a problem domain make it reasonable to assume that the posterior can be approximated by a specific distribution, such as a Gaussian, then a particle filter or RBPF may not be the best solution.

Another solution is to avoid sampling from high-dimensional state spaces. One possible way to do this is to factor a large state space into independent low-dimensional components. The work presented by Montemerlo and colleagues showed that they could simultaneously estimate a robot's position and the location of 50,000 landmarks [18]. This was possible because the parameters for each of the landmarks were independent from each other. In this case, the size of the state space only grows linearly with the number of parameters being estimated rather than exponentially. Alternatively, one could use a low dimensional parameterization of the state space to approximate the full state space. For example, the space of all possible strategies in Texas Hold'em poker is far too large to model. Instead, we could use a low dimensional parameterization of agent behaviour that describes a likely subspace of opponent strategies. We take this approach in this thesis by using the parameterization of undominated Kuhn poker in place of the full Kuhn poker parameterization. If a reasonable parameterization exists in the problem domain, we can use particle filters and RBPFs to model agents in the low dimensional parameterization rather than the full state space.

6.1.2 Using the Model in Larger Domains

Aside from the problems in forming an accurate model of an agent, there is also the question of what to do with the resulting model. In Kuhn poker we can quickly compute our greedy best response to a given model. In other domains it is not clear what to do with the model because computing a response may not be tractable in real time. This problem is not specific to our agent modelling techniques, since any modelling technique will need to address this problem, but it is still an issue that will need to be solved when attempting to model agents in domains that are more complex than Kuhn poker.

One possible solution to this problem comes from Johanson and colleagues' recent progress on computing robust counter-strategies [15]. This work examines how to compute *restricted Nash responses*. These responses are designed to be near equilibrium strategies that exploit other agents while being, themselves, difficult to exploit. This technique makes the resulting responses robust to errors in the agent model. The authors have already demonstrated the approach in Texas Hold'em poker and their experiments show that the resulting strategies are much more robust than previous best response techniques. Unfortunately, this technique uses linear programming to generate a restricted Nash response and it cannot be run in real time on a domain as large as Texas Hold'em poker. Despite this drawback, the technique is still useful offline for precomputing robust counter-strategies in Texas Hold'em poker.

6.1.3 Motion Models and Agent Dynamics

As we showed in our results, simple motion models may be sufficient to achieve good results. Unfortunately, it is not clear that this will be the case when attempting to model more complex dynamics. We could improve the modelling agent's performance if its motion model was more representative of an agent's true dynamics. In our domain of Kuhn poker, creating some simple non-oblivious motion models would be an interesting next step that may yield improvements when playing non-oblivious opponents. In general, non-oblivious models would be an asset when modelling humans because oblivious models are unlikely to capture a human's adaptability. Unfortunately, creating more representative models of agent dynamics requires some knowledge about how agents change their behaviour over time. Even in our small domain of Kuhn poker, this is difficult to describe mathematically.

One way to create motion models that more accurately capture true dynamics is to learn the models from samples of agents' behaviours. We may be able to extract general trends in agent dynamics from such sample data. Using the extracted models in a RBPF could allow us to learn an agent's specific model parameters online throughout our interactions with them. Texas Hold'em poker, for example, is an ideal domain for this approach since it is popular enough that there is no shortage of human players willing to play the game and generate the necessary samples. Other domains may not be able to gather samples of agent dynamics so easily.

In general, we also need to ensure that the motion models we create are computationally efficient. Creating sophisticated motion models that are also efficient may be challenging since their computational cost relies on both the complexity of the domain and the complexity of the dynamics one intends to model. Even if we could learn an agent's true dynamics, we may need to settle for an efficient approximation of the dynamics if the true dynamics are too costly to duplicate.

6.1.4 Efficient Observation Models

Finally, we consider one subtlety in obtaining the models necessary for state estimation. Because of Kuhn poker's small size, it is computationally inexpensive to determine observation probabilities. In more complex domains the cost of the observation model may become a problem – especially if real-time modelling is required. Of course, a larger domain does not necessarily mean this will be a problem. For instance, in Texas Hold'em poker it has been shown that we can compute observation probabilities in real-time [23].

6.2 Future Work

Despite our focus on the small domain of Kuhn poker, we believe that state estimation techniques can be effectively applied to much larger domains. Future directions for this work consist of attempting some of the aforementioned solutions so we can use state estimation algorithms for agent modelling in new domains.

One area of future work is extending our state estimation techniques to modelling opponents in full Texas Hold'em. Although the size of the Texas Hold'em game tree makes some computations prohibitive, previous work provides us with some of the pieces we need to apply our state estimation techniques to this very complex domain. We already have a considerable amount of data for learning a human dynamics model. Furthermore, an observation model already exists [23]. Of course, overcoming the remaining challenges described in Section 6.1 will not be trivial. For instance, even after we learn a human dynamics model, evaluating its effectiveness and robustness against humans will be a difficult task.

Finally, having only applied state estimation to agent modelling in Kuhn poker, it would be interesting to apply these techniques for agent modelling in other domains – games or otherwise. One possible application is to use state estimation to model biological processes such as the response of viruses and infections to different medical treatments. More generally, there are potential applications for agent modelling throughout the world. Applying state estimation to any of these domains would provide more insight into the limitations of using state estimation for modelling agents. Good performance in other domains would also provide further evidence that these techniques have a place in the agent modelling community.

6.3 Concluding Remarks

The pervasiveness of computers in our everyday lives has resulted in an increasing number of applications that require interaction between independent agents – especially between a human and a computer. The lack of effective modelling and inference techniques in these applications is becoming more apparent. The problem of modelling agents' goals, behaviour, and beliefs is known to be very challenging. Although research throughout the agent modelling community has yielded many approaches to the problem, state estimation has been largely overlooked.

This thesis presented particle filters and Rao-Blackwellized particle filters as two state estimation algorithms and demonstrated their potential in the domain of Kuhn poker. Poker is a challenging domain in artificial intelligence. The combination of hidden information, stochastic events and choices, and dynamic behaviour presents many challenges to creating a good solution. Our work, though only demonstrated in the small domain of Kuhn poker, attempts to address these challenges in a robust and principled way. Our results show that state estimation algorithms have a place in the agent modelling literature. Furthermore, the extensive research on state estimation algorithms in other scientific communities (*e.g.* robotics) is a valuable resource that should be tapped for modelling agents. It is our hope that this work will spark future research in using state estimation algorithms for agent modelling problems.

Bibliography

[1] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 322–331. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[2] Curt Bererton. State estimation for game AI using particle filters. In *AAAI workshop on challenges in game AI*, 2004.

[3] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 661–668, 2003.

[4] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, 2002.

[5] Darse Billings, Aaron Davidson, Terence Schauenberg, Neil Burch, Michael Bowling, Robert Holte, Jonathan Schaeffer, and Duane Szafron. Game tree search with adaptation in stochastic imperfect information games. In Yngvi Bjornsson, Nathan Netanyahu, and Jaap van den Herik, editors, *Computers and Games*, pages 21–34. Springer-Verlag, 2004.

[6] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 493–498. AAAI Press, 1998.

[7] Hung H. Bui, Svetha Venkatesh, and Geoff West. Policy recognition in the abstract hidden Markov model. *Journal of Artificial Intelligence Research*, 17:451–499, 2002.

[8] David Carmel and Shaul Markovitch. Learning models of intelligent agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 62–67. AAAI Press, 1996.

[9] Aaron Davidson, Darse Billings, Jonathan Schaeffer, and Duane Szafron. Improved opponent modeling in poker. In *Proceedings of the 2000 International Conference on Artificial Intelligence*, pages 1467–1473, 2000.

[10] Andrew Gilpin and Tuomas Sandholm. A competitive Texas hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, 2006.

[11] Kwun Han and Manuela Veloso. Automated robot behavior recognition applied to robotic soccer. In *Proceedings of the International Symposium of Robotics Research*, pages 199–204, October 1999.

[12] Bret Hoehn. The effectiveness of opponent modelling in a small imperfect information game. Master's thesis, University of Alberta, 2006.

[13] Bret Hoehn, Finnegan Southey, Robert C. Holte, and Valeriy Bulitko. Effective short-term opponent exploitation in simplified poker. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 783–788, 2005.

[14] Michael Isard and Andrew Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[15] Michael Johanson, Martin Zinkevich, and Michael Bowling. Computing robust counter-strategies. In *Advances in Neural Information Processing Systems 20*. MIT Press, 2008.

[16] H. W. Kuhn. A simplified two-person poker. *Contributions to the Theory of Games*, 1:97–103, 1950.

[17] Jane Liu and Mike West. Combined parameter and state estimation in simulation-based filtering. In Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors, *Sequential Monte Carlo Methods in Practice*, pages 197–224. Springer-Verlag, New York, 2000.

[18] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 593–598. AAAI Press, 2002.

[19] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670, 2000.

[20] Guillermo Owen. *Game Theory*. Academic Press, third edition, 1995.

[21] Bob Price and Craig Boutilier. A Bayesian approach to imitation in reinforcement learning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 712–720, 2003.

[22] Terence Schauenberg. Opponent modelling and search in poker. Master's thesis, University of Alberta, 2006.

[23] Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: Opponent modelling in poker. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 550–558, 2005.

[24] Geir Storvik. Particle filters for state-space models with the presence of unknown static parameters. *IEEE Transactions on Signal Processing*, 50:281–289, February 2002.

[25] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[26] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.