

Closed-form Solutions to Sequential Decision Making within Markets

Shamin Kinathil

A thesis submitted for the degree of
Doctor of Philosophy
The Australian National University

November 2019

© Shamin Kinathil 2018
All Rights Reserved

Declaration

I hereby declare that this thesis is my original work which has been done in collaboration with other researchers. This document has not been submitted for any other degree or award in any other university or educational institution. The following papers are accepted for publication in peer reviewed conference proceedings listed in a reverse chronological order:

- **(Chapter 7)** Kinathil, S., Soh, H., and Sanner, S. Analytic Decision Analysis via Symbolic Dynamic Programming for Parameterized Hybrid MDPs. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017. (2017), pp. 181–185.
- **(Chapter 5)** Kinathil, S., Sanner, S., Das, S., and Penna, N. D. A Symbolic Closed-Form Solution to Sequential Market Making with Inventory. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI 2016, New York, NY, USA, 9-15 July 2016 (2016), pp. 3609–3615.
- **(Chapter 6)** Kinathil, S., Sanner, S., and Penna, N. D. Closed-form Solutions to a Subclass of Continuous Stochastic Games via Symbolic Dynamic Programming. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014 (2014), pp. 390–399.

To Georgia Miriam Neeson.

Acknowledgements

Firstly, I would like to express my deepest gratitude to my supervisor, Dr. Scott Sanner, for all his guidance, patience and support throughout my doctoral study. Scott's immense knowledge and enthusiasm for research helped me to stay motivated. I would also like to thank Scott and his family for the lovely home cooked meals in Canberra and Corvallis.

My sincere thanks also goes to Professor Sylvie Thiébaux and Professor Richard Nock for their words of encouragement and support. I would also like to thank my co-authors Nicolás Della Penna and Harold Soh for exposing me to different perspectives on my doctoral research.

I acknowledge the financial, academic and technical support provided by the National ICT of Australia (NICTA), Data61 and the Australian National University (ANU) for my doctoral program.

I am also indebted to my colleagues at NICTA, Data61 and the ANU for providing a friendly environment that allowed me to grow as researcher. My special thanks to Suvash Sedhain, Shahin Namin, Hadi Afshar, Kar Wai Lim, Paul Rivera, Suranga Seneviratne, Paul Rimba and Zac Cranko. Thank you for the stimulating discussions over games of foosball, lunch and coffee.

Finally, and most importantly, I would like to thank Georgia Neeson. I could not have done this without her love, support and encouragement. I dedicate this thesis to her.

Abstract

Sequential decision making is a pervasive and inescapable requirement of every day life. Deciding upon which sequence of actions to take is complicated by incomplete information about the environment, the effects of each decision upon the future state of the environment, ill-defined objectives and our own cognitive limitations. These challenges are exacerbated in financial markets which are in a constant state of flux, with prices adjusting to new information, winning traders replacing losing traders and the introduction of new technologies. Decision theoretic planning provides powerful and flexible frameworks for many sequential decision making problems within financial markets. Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) are two such frameworks, which can model financial decision making problems with either or both discrete and continuous variables. Traditional methods to solve MDPs and POMDPs use Monte-Carlo based methods or discretise the continuous state space. This is often to the detriment of the quality of the solution, which in many financial domains, such as those involving asset prices, are required to be exact and closed-form. Closed-form solutions, by virtue of being declarative and symbolic, allow the decision maker to undertake further analysis and optimisation. In this thesis we present novel techniques to calculate exact and closed-form solutions to new classes of MDPs and POMDPs by leveraging Symbolic Dynamic Programming (SDP). Our specific contributions include: (i) closed-form solutions to a subclass of Continuous Stochastic Games, which we use to encapsulate and solve a zero-sum game between a Binary Option trader and an adversarial market; (ii) closed-form solutions to Sequential Market Making with Inventory, which extends seminal works from market microstructure theory and algorithmic market-making; and (iii) Analytic Decision Analysis for Parameterised Hybrid MDPs, which we use to examine the sensitivity of a model for optimal transaction execution to its parameters. Our experimental evaluations confirm the efficacy of our novel solutions for a range of new classes of MDPs and POMDPs.

Contents

Abstract	xi
1 Introduction	1
1.1 Financial Markets	1
1.2 Formal Models of Decision Making	3
1.3 Decision Theoretic Planning	5
1.4 Motivating Examples	6
1.4.1 Market Making	6
1.4.2 Optimal Execution	7
1.4.3 Trading Agents	7
1.5 Contributions	8
1.6 Outline	9
2 Frameworks for Sequential Decision Making	11
2.1 Markov Decision Processes	12
2.1.1 The Markov Decision Process Framework	12
2.1.2 Solution Concepts	13
2.1.3 MDPs for Financial Markets	16
2.2 Partially Observable Markov Decision Processes	17
2.2.1 The POMDP Framework	17
2.2.2 Solution Concepts	18
2.2.3 POMDPs for Financial Markets	23
2.3 Other Decision Making Frameworks	24
2.3.1 Multi-Armed Bandits	24
2.3.2 Multi-Agent Frameworks	25
2.4 Summary	26
3 Exact Discrete Solutions	29
3.1 Markov Decision Processes	29
3.1.1 Discrete $\mathcal{S}$	30
3.1.2 Continuous Extensions	33
3.2 Partially Observable Markov Decision Processes	33
3.2.1 Discrete $\mathcal{S}$	34
3.2.2 Continuous Extensions	37
3.3 Local Solution Techniques	39
3.3.1 Markov Decision Processes	39
3.3.2 Partially Observed Markov Decision Processes	40

3.4	Summary	41
4	Structured Discrete and Continuous Solutions	43
4.1	Structured MDPs and POMDPs	43
4.1.1	Structured Representations	44
4.1.2	Structured Value Iteration	46
4.2	Symbolic Dynamic programming	46
4.2.1	Case Statements	46
4.2.2	Case Operations	47
4.2.3	SDP for Financial Markets	51
4.3	Structured Solutions	51
4.3.1	Decision Trees and Algebraic Decision Diagrams	52
4.3.2	Extended Algebraic Decision Diagrams	53
4.3.3	XADD Operations	55
4.4	Summary	55
5	Closed-form solutions to Sequential Market Making with Inventory	57
5.1	Optimal Sequential Market Making Model	57
5.1.1	Solution Techniques	61
5.2	SDP for Continuous State POMDPs	61
5.3	Experimental Results	63
5.3.1	Sequential Optimality	64
5.3.2	Inventory Sensitivity	64
5.3.3	Trader Models	66
5.4	Related Work	66
5.5	Summary	68
6	Closed-form solutions to a Subclass of Continuous Stochastic Games	69
6.1	Zero-sum Continuous Stochastic Games	69
6.1.1	Solution Techniques	70
6.2	SDP for Zero-sum Continuous Stochastic Games	72
6.3	Experimental Results	73
6.3.1	Continuous Stochastic Matching Pennies	74
6.3.2	Binary Option Stochastic Game	77
6.3.3	Robust Energy Production	79
6.4	Related Work	81
6.5	Summary	81
7	Analytic Decision Analysis for Parameterised Hybrid MDPs	83
7.1	Parameterised Hybrid MDPs	83
7.1.1	Inverse Learning for Multi-objective PHMDPs	84
7.1.2	Sensitivity Analysis for PHMDPs	85
7.1.3	Nonlinear Parameterised Policy Optimization Methods for PH-MDPs	85
7.1.4	Solution Techniques	85

7.2	SDP for Parameterised Hybrid MDPs	85
7.2.1	Inverse Learning for Multi-objective PHMDPs	86
7.2.2	Sensitivity Analysis for PHMDPs	86
7.2.3	Nonlinear Parameterised Policy Optimization Methods for PH- MDPs	87
7.3	Experimental Results	87
7.3.1	Inverse Learning for Multi-objective Navigation	87
7.3.2	Influenza Public Health Policy	89
7.3.3	Optimal Execution	89
7.4	Related Work	91
7.5	Summary	93
8	Conclusions	95
8.1	Summary of Contributions	95
8.2	Future Work	96
8.2.1	Sequential Decision Making Frameworks	96
8.2.2	Solution Algorithms	97
8.2.3	SDP Data Structures	98
	References	99

List of Figures

1.1	Risk Utility Functions. The concavity of the utility functions indicate the marginal utility gained as the value increases.	4
1.2	Prospect Utility Function under the Allias paradox. The utility function under the Allias paradox is steeper for losses than it is for gains.	4
2.1	Graphical Model representation of an MDP. At every epoch $d \in \mathcal{D}$, action $a_d \in \mathcal{A}_d$ is executed in the current state s_d . This affects the transition to the next state s_{d+1} , while also obtaining the reward r_d	12
2.2	Graphical Model representation of a POMDP. At every epoch $d \in \mathcal{D}$, the agent perceives observation $o_d \in \mathcal{O}_d$ but does not know the underlying state $s_d \in \mathcal{S}_d$. Action $a_d \in \mathcal{A}_d$ is executed in the current state s_d , which affects the transition to the next state s_{d+1} while also obtaining the reward r_d	18
2.3	A policy tree for epoch d . When in a node, the associated action is applied and then, according to the observation received, the next node is reached by descending the branch labelled by the observation.	20
2.4	Five α -vectors representing the Value function of a 2 state POMDP. The belief space is on the x-axis and the value is on the y-axis. The linear segments that constitute the value function are in bold.	22
3.1	An α -function defined with respect to a Uniform belief. The upper surface of the plot indicates the optimal α -function for the value belief defined by $v \sim \mathcal{U}[m - w, m + w]$, for a given mean m and width w	38
4.1	DBN representation for the transition and reward functions of a simplified factored Mars Rover. For any action a , a state variable $\mathcal{X}_i$ is conditionally independent of its non-descendents given its parents $\text{Pa}(\mathcal{X}_i)$	45
4.2	Compact tree-based representations of a Boolean function $f : \mathbb{B}^3 \rightarrow \mathbb{R}$ defined in Table (4.2). The Decision Tree exploits context-specific independence, whereas the Algebraic Decision Diagram exploits context-specific independence and shared substructure.	53
4.3	A plot of a function $f(x, y, a = \text{true})$ for continuous variables x and y and boolean variable a ; and its XADD representation.	54

5.1	The optimal Sequential Market Making POMDP. The model consists of variables v - value, i - inventory, actions b - <i>bid</i> , a - <i>ask</i> , intermediate variables t and u , observation - o , and reward - R . Primed variables represent the “next state” of the variable. Arrows represent a dependency.	58
5.2	Sequentially optimal $\mathcal{H} = 3$ policy. Nodes represent actions $a \in \mathcal{A}$ and arrows represent an observation $o \in \mathcal{O}$	64
5.3	Optimal $\mathcal{H} = 3$ <i>bid</i> and <i>ask</i> prices under different market settings. The initial inventory level $i \in \{0, 1, 2\}$. (bid_1, ask_1) and (bid_2, ask_2) are the optimal prices for a trader population comprising of 50% and 10% uninformed traders, respectively.	65
5.4	Optimal $\mathcal{H} = 3$ α -functions for $i = 1.0$ under different trader models. The upper surface of the plot indicates the optimal α -function for the market-maker’s value belief defined by $v \sim \mathcal{U}[m - w, m + w]$, for a given mean m and width w	67
6.1	The optimal value functions of the continuous stochastic matching pennies game for Player 1 at horizon 4 under (a) symmetric and (b) asymmetric reward structures. Threshold values are set to $c = 0.3$ and $d = 0.7$ and are highlighted in red and green, respectively. The step size is $k = 0.3$	76
6.2	The optimal value function of the binary option stochastic game for the trader at horizon 20. The strike price is set to $\kappa = 45.0$ and the increment and decrement values are set to $u = 1.0$ and $d = 1.0$, respectively. Under the domain specification the value function is invariant to the inventory i	78
6.3	The optimal value function of the robust energy production game for the producer at horizon 8. The production and demand increase and decrease variables were set to $prd_u = prd_d = 1.0$ and $nat_u = nat_d = 0.5$, respectively.	80
7.1	Value function and Nonlinear optimization for the Multi-objective Navigation domain.	88
7.2	Value function and Nonlinear optimization for the Influenza Public Health Policy domain.	90
7.3	Value function and Nonlinear optimization for the Optimal Execution domain.	92

List of Tables

2.1	MDP Policy Families.	14
2.2	MDP Performance Criteria.	14
4.1	Symbolic Functions used in this Thesis.	47
4.2	Tabular representation of a Boolean function $f : \mathbb{B}^3 \rightarrow \mathbb{R}$	53
6.1	Symmetric reward structure for Player 1.	75
6.2	Asymmetric reward structure for Player 1.	77

Introduction

Decision making is a pervasive and inescapable requirement of every day life. Each decision we make is made with varying amounts of information and uncertainty about its outcomes. Confronting uncertainty is not easy, we naturally try to avoid it, sometimes even pretending that it does not exist, and when coupled with our cognitive limitations [Sweller, 1988, Simon, 1991] can result in outcomes that have far reaching adverse consequences for individuals and societies.

Despite this, our longevity as a species is a testament to our ability to make decisions with good outcomes. Our primitive ancestors turned to priests and oracles to reveal the uncertain future. The methods may have changed, but our desire for predictions of the future still abound. To deal with these problems on a rational basis, we have developed a theoretical structure for decision making that includes uncertainty.

In the first section we begin by introducing financial markets, a setting where much of the early work in *risk* and *uncertainty* were undertaken, and in which *trillions* of dollars are transacted daily. Financial decisions are among the most important decisions that people make, yet a lack of information and cognitive constraints lead investors to make financially unsound decisions [Frydman and Camerer, 2016]. Financial market crashes, such as the Global Financial Crisis (GFC) of 2007-2008, are an example of extreme repercussions of unsound decisions in financial markets [Barro and Ursúa, 2017].

1.1 Financial Markets

Financial markets facilitate the exchange of assets, both tangible and financial. The most important function of markets is to facilitate the recycling of funds, in the form of assets, from surplus agents, also known as sellers, to deficit agents, known as buyers, in an efficient manner. Surplus agents are made up of individuals, companies and governments that have surplus funds that they wish to invest. In contrast, deficit agents are made up of individuals, companies and government bodies that do not have sufficient current funds themselves. There is considerable heterogeneity amongst surplus and deficit agents in terms of appetite for risk as well as time preferences. Financial intermediaries design products and securities that facilitate the

exchange of funds between surplus and deficit agents.

A financial security is a saleable right to receive a sequence of future payments. The size of each future payment can be either guaranteed or determined by the outcome of some uncertain future event. The number of payments in the sequence can be finite, or continue indefinitely into the future. Three such financial securities include bonds, common stocks and call options.

A bond is a saleable right to receive a finite sequence of guaranteed payments. Bonds are contracts between borrowers and lenders, the former of which issues the bond and the latter of which owns the bond. Issuers of bonds typically do so to fund the purchase of an asset. A bond is also saleable which means that the investor can sell the bond to another investor any time before the last payment is due. A common stock is a saleable right to receive an indefinitely long sequence of future payments, with the size of each payment contingent on both the firm's future earnings. Unlike a bond, a share of common stock is not a contractual right to receive a sequence of payments. Payments depend on the ability of the firm to generate earnings. Firms issue common stock to raise money for projects. Unlike the case of a bond, the investors who purchase newly issued shares of common stock from a firm do not have the right to get their money back at the end of some specified number of periods. A call option is the right, but not the obligation, to purchase a specific security at a fixed price by paying an exercise price prior to a fixed date. The price of the option fluctuates over the life of the option according to the investor's beliefs about the future values of the underlying security.

In addition to providing a mechanism by which financial securities are exchanged, markets also provide on a continuous basis, a price at which buyers and sellers are happy to conduct trade. The price reflects the laws of supply and demand: if buyers exceed sellers of an asset, this will push it up in price, whereas if sellers exceed buyers the price will be driven down.

The structure of a financial market indicates who can access the trading venue, which instruments can be traded, the location of the trading sessions and the order routing systems. The market structure also stipulates the transparency of pre- and post-trade information that market participants can access. As a result, the market structure has a significant impact on the decision making process of its participants.

Markets are in a constant state of flux, with prices adjusting to new information as winning traders replace losing traders and as new technologies evolve. Highly skilled traders employ clever strategies in their search for trading profits and ultimately profit from investors, gamblers and foolish traders. The decision making abilities of traders is significantly impacted by the availability of information and their ability to quantify and mitigate uncertainties in the market. The sources of uncertainties in financial markets can be classified in three categories [Beckert and Berghoff, 2013]:

1. Strategic uncertainty, where the success or failure of strategies depend on the actions of other investors pursuing their own interests;
2. Uncertainty originating from contingent assessments of the value of goods in

the market; and

3. Uncertainty stemming from events occurring in the future that are unknown to actors in the present due to the openness of the future.

While uncertainty creates problems for the coordination of exchange and causes insecurity for investors, it is also a crucial source of opportunities for profit. Hence, investors in financial markets must make decisions that depend on an uncertain future. In this thesis we introduce novel sequential decision making frameworks and algorithms that allow a financial market investor to calculate optimal decisions in the presence of uncertainty. In the proceeding section we introduce formal models of decision making from economics and artificial intelligence.

1.2 Formal Models of Decision Making

The prevailing framework of human decision making within economics and finance is that of Expected Utility Theory (EUT) [Neumann and Morgenstern, 1944]. Under EUT, all decisions under risk and uncertainty are made on the basis of preferences and utility axioms. The four axioms that define a *rational* decision maker are: (i) completeness; (ii) transitivity; (iii) independence; and (iv) continuity. The axioms of rational decision making, based on normative theories, imply that people would always choose the best option, having first evaluated all alternatives.

EUT permits a wide range of preferences under risk. A risk-averse decision maker can be represented by a concave utility function, a risk-neutral decision maker by a linear utility function and a risk-seeking decision maker by a convex utility function. The three utility functions are shown in Figure (1.1). The concavity of the utility functions indicate the *marginal* utility gained as the value increases. In Figure (1.1a) the marginal utility decreases as the value increases, in Figure (1.1b) the marginal utility remains constant as the value increases, whereas in Figure (1.1c) the marginal utility increases as value increases. As an example, consider an investor given the choice between receiving a guaranteed \$50.00 versus receiving \$100.00 on the basis of the flip of a fair coin. The expected reward in this lottery is \$50.00. A risk-averse investor is willing to accept an amount less than the expected amount of the lottery, that is, they would accept the guaranteed \$50.00. A risk-neutral investor simply aims to maximise their expected value, as a result they would be indifferent between the two scenarios. Finally, a risk-seeking investor is only willing to accept an amount greater than the expected amount of the lottery and, as a result, would accept the reward associated with the outcome of the coin flip.

Despite the elegance of the EUT framework many experimental studies have shown that decision makers often systematically violate the utility axioms. Recent research has shown that humans have limited processing and memory capacity, making it impossible to carry out the mental activities underpinning rational decision making. Alternate theories of human decision making, such as prospect theory, seek to address the drawbacks of the EUT framework.

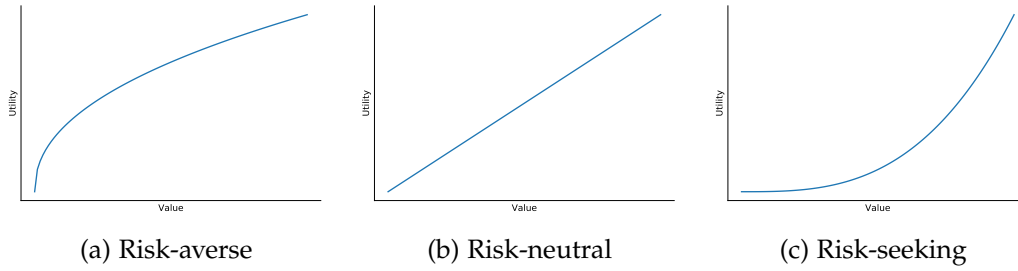


Figure 1.1: Risk Utility Functions. The concavity of the utility functions indicate the marginal utility gained as the value increases.

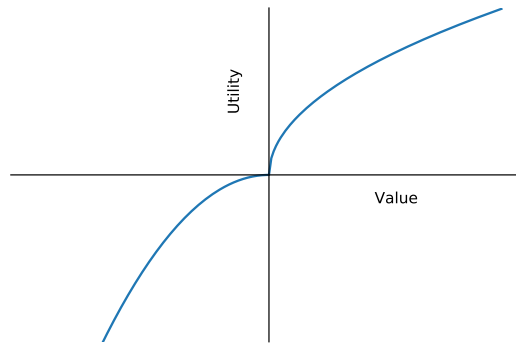


Figure 1.2: Prospect Utility Function under the Allias paradox. The utility function under the Allias paradox is steeper for losses than it is for gains.

Prospect theory (PT) approaches human decision making from the perspective of behavioural science. The main tenets of the theory, discovered through experimentation, have been used to address the paradoxes and problems of EUT [Tversky and Kahneman, 1974]. One such paradox, the Allias paradox [Allais, 1953], showed that when decision making situations involve possible gains and losses, people's preferences over negative outcomes are not symmetric with respect to their preferences over positive outcomes. In other words, people are risk-averse over outcomes involving gains and risk-seeking over outcomes involving losses. The Allias paradox violates the independence axiom of EUT. Figure (1.2) shows that the utility function under the Allias paradox is steeper for losses than it is for gains.

Further theories from behavioural science have shown that human decision makers have evolved specific cognitive biases and heuristics that make complicated problems more tractable, but lead to errors. These heuristics, known as System 1 thinking, are in stark contrast to the controlled, rule-based approach of rational decision making, often described as System 2 [Kahneman and Frederick, 2002]. System 1 aims to explain how humans actually make decisions and why these are often subject to error and bias. Sub-optimal decisions made as a result of these biases can have important consequences for societies, politics and financial markets [De Bondt and Thaler, 1985].

The insights of behavioural economics and psychology indicate that human decision making may be substantially improved when assisted by computer based decision theoretic systems. Decision theoretic systems provide a formal framework to not only express uncertainties and utilities but also leverage computational power to mitigate the bounded rationality of human decision makers. In the next section we introduce one such computational framework that can be used to encode varying notions of utility, objective and risk, and most importantly, calculate optimal decisions. In this thesis we utilise Expected Utility Theory which, despite its drawbacks, is a useful approximation of rational human behaviour in financial markets.

1.3 Decision Theoretic Planning

Decision Theoretic Planning (DTP) [Boutilier et al., 1999] aims to determine an optimal sequence of actions in the presence of incomplete information, uncertain outcomes and potentially ill-defined goals. As such, DTP, readily lends itself to optimal decision making in complex environments.

Decision Theoretic Planning draws upon the fields of control theory and operations research. Control theory aims to manipulate inputs into a deterministic or stochastic system to obtain a desired effect on the output of the system. Operations research aim to provide rational bases for decision making by seeking to understand and structure complex situations and to use this understanding to predict system behaviour and improve system performance. The seemingly disparate fields of control theory and operations research converge within decision theoretic planning, where models such as Markov Decision Processes (MDPs) [Puterman, 1994] and Partially Observable Markov Decision Processes (POMDPs) [Kaelbling et al., 1998] are frequently employed. MDPs provide a framework for modelling complex control problems with complete observability, uncertain action effects and a utility based specification of reward. POMDPs provide a natural extension to this framework by allowing for partial observability, and incomplete knowledge of the environment dynamics. The MDP and POMDP frameworks have also gained in popularity within Artificial Intelligence as a model for planning under uncertainty [Bertsekas, 1987, Boutilier et al., 1999].

The utility of DTP within financial markets is constrained by the ability of its models to represent problems involving continuous states, for example those involving price, which is most naturally represented by a real number, and expressive representations of utilities and rewards. Applications of MDPs and POMDPs within such domains often discretise the state space, an approach which leads to the curse of dimensionality [Bellman, 1957], in which the computational cost required to solve such formulations increases with the size of the state space. An obvious alternative is to avoid discretisation altogether by opting for solutions directly applicable to continuous state problems. In this thesis we address how to leverage DTP models such as the MDP and POMDP to solve financial market problems with continuous state and expressive representations of utilities and rewards.

1.4 Motivating Examples

To motivate the remainder of this thesis, we provide three examples of decision making problems found in financial markets: (i) market-making; (ii) optimal execution; and (iii) trading agents. We note that the examples are not completely isolated from one another, that is, a market-maker may be implemented as a trading agent and may also choose to solve the optimal execution problem. We later show how these problems can be formulated and solved using novel techniques introduced in this thesis.

1.4.1 Market Making

Market-makers serve an important liquidity-provision role in many financial markets by providing immediacy to transactions. They do this by always taking the other side of any transaction, at some price. This price may either be regulated, as used to be the case in the NYSE, arise through competition, as in the NASDAQ [Schwartz, 1991], or through algorithmic design, as in many prediction markets [Brahma et al., 2012].

In practice market-makers often operate in tandem with a Continuous Double Auction market structure in which both buyers and sellers submit limit orders in the form of bids and asks, which are then matched algorithmically. Theoretical models of market making, which we address in this thesis, typically abstract away limit orders from other traders and model the market as a *dealer market*, with the market-maker taking one side of every order. In these models it is common for the market-maker to quote a *bid* price, a price at which they are willing to buy some number of shares, and an *ask* price the price at which they are willing to sell some number of shares, at any point in time. The difference between the two prices, known as the bid-ask spread, serves three main purposes:

1. To provide an incremental profit to motivate the market-maker to actually provide their services;
2. To compensate the market-maker for the risk inherent in holding inventory [Amihud and Mendelson, 1980]; and
3. To compensate the market-maker for the adverse selection encountered in trading with a potentially more informed population [Glosten and Milgrom, 1985].

When setting the *bid* and *ask* price, market-makers must trade-off between setting prices to extract maximum profit from the next trade, versus setting prices to get as much information about the new value of the asset so as to generate larger profits from future trades.

The market-making problem naturally lends itself to a decision theoretic planning formulation. The sequential nature of the decision making process, whereby the prices set at one period of time affects future profits, and the fact that price is most naturally represented by a continuous variable, motivates the use of the novel techniques presented within this thesis.

1.4.2 Optimal Execution

One of the most important challenges faced by participants in financial markets is how to sell assets to move a portfolio from one position to another within a specified amount of time. The liquidation of a large number of assets is a challenge due to two factors:

1. Possible lack of a counterparty; and
2. Price impact that depresses prices by increasing supply.

Both factors can lead to a decline in price. Counterparty risk exists because a large sell order may exhaust all current buyers, bringing about a decline in price. Price impact occurs as market participants begin to detect the total volume being traded and consequently adjust their buying price downwards to anticipate order matching. Therefore, it is important to execute a strategy in a way that minimises the effect of the aforementioned factors.

While there is no formal definition of optimal execution, a commonly used measure is *implementation shortfall* [Perold, 1988]. Implementation shortfall measures the difference between the theoretical cost of transacting the portfolio, which does not include transaction costs, and the real portfolio, which does. The shortfall can be divided into two components:

1. Explicit components due to broker fees and taxes; and
2. Implicit component due to market impact cost, opportunity cost and bid-ask spread.

An optimal execution algorithm must solve three problems:

1. When to trade, in what size and for roughly how long;
2. Decide whether to place market or limit orders and at what price level(s); and
3. Given a limit or market order, to which venue should the order be sent. A limit order is a buy or sell at a specific price, whereas a market order is a buy or sell at the current price.

The aim of an optimal execution algorithm is to minimise the shortfall while maximising profit. Optimal execution can be formulated as a decision theoretic planning problem with *continuous* states and a utility function that embodies the notion of implementation shortfall. This thesis introduces techniques that allow the aforementioned formulation of this problem to be solved exactly and in closed-form.

1.4.3 Trading Agents

Trading agents are automated strategies used for trading in financial markets. They operate with minimal human intervention and can range from simple rule-based systems to more sophisticated algorithms based on machine learning and game theory. The use of trading agents within financial markets are driven by three main factors:

-
1. The need to respond to information at a rate that is much faster than humanly possible;
 2. The ability of trading agents to act autonomously and adapt to changing market conditions; and
 3. The ability for trading agents to be replicated and operate at scale over many different assets and markets.

Trading agents are most often used to catch opportunistic profits in the form of arbitrage opportunities. An arbitrage opportunity arises when assets are simultaneously bought and sold in markets where there are price differences. The arbitrageur, in this case the trading agent, profits from the difference in price. The lure of guaranteed profits has led to an arms race amongst financial companies, each aiming to have the fastest trading agent [Johnson et al., 2013].

A trading agent can be modelled using the decision theoretic planning framework. The continuous nature of the state space, namely price, lends itself to being solved using the techniques introduced in this thesis.

1.5 Contributions

The main contributions of the thesis are summarised below:

1. **Closed-form solutions to a Subclass of Continuous Stochastic Games.** Zero-sum stochastic games provide a formalism to study competitive sequential interactions between two agents with diametrically opposing goals and evolving state. While a solution to such games with discrete state was presented by Littman [Littman, 1994], the continuous state version of this game remains unsolved. In many instances continuous state solutions require nonlinear optimisation, a problem for which closed-form solutions are generally unavailable. We present an exact closed-form solution to a sub-class of zero-sum continuous stochastic games that can be solved as a parameterised linear program by utilising symbolic dynamic programming. The efficacy of the novel framework is used to calculate an optimal solution to a zero-sum game between a binary option trading agent and an adversarial market.
2. **Closed-form solutions to Sequential Market Making with Inventory.** We are the first to address the open question of sequentially optimal market-making with inventory based reasoning by extending the seminal work of [Glosten and Milgrom, 1985], [Amihud and Mendelson, 1980] and [Das and Magdon-Ismail, 2009]. We introduce a novel continuous state POMDP framework that is the first to solve, exactly and in closed-form, the optimal market-making problem with inventory, fixed asset value, arbitrary belief state priors, trader models and reward functions.
3. **Analytic Decision Analysis for Parameterised Hybrid MDPs.** We present Parameterised Hybrid MDPs as a unifying framework that enables the inverse

learning of parameters of multi-objective rewards, parameter sensitivity analysis and non-convex analysis and optimization of policy parameters. We provide an algorithm that solves this class of PHMDPs exactly and in closed-form by defining a parameterised variant of Symbolic Dynamic Programming (SDP) [Boutilier et al., 2001] extended to hybrid MDPs [Sanner et al., 2011]. We use the PHMDP framework in conjunction with parameterised SDP and state-of-the-art non-convex optimizers to calculate the first exact solutions to: (i) inverse learning of the parameters of a multi-objective reward domain; (ii) non-convex optimization of public health policies in epidemic models; and (iii) exact sensitivity analyses of trading strategies for portfolio transactions.

1.6 Outline

This chapter introduced our research problem: exact solutions to classes of MDPs, and POMDPs and their applications within markets. The remainder of this thesis is structured as follows.

Chapter 2 begins by introducing the sequential decision making problem. We then discuss two frameworks and solution concepts for sequential decision making and illustrate how they can be applied within financial markets. We conclude the chapter by introducing alternate frameworks for decision making.

Chapter 3 presents algorithms and techniques to calculate exact global solutions to the Markov Decision Process and Partially Observable Markov Decision Process. We begin by presenting canonical algorithms to solve discrete state and action formulations of MDPs and, discrete observations, in the case of POMDPs. We conclude the chapter by introducing local solution methods for MDPs and POMDPs.

Chapter 4 introduces structured discrete and continuous solution algorithms for MDPs and POMDPs. Symbolic Dynamic Programming, a process of performing dynamic programming by symbolic manipulation, is introduced as a method to solve MDPs and POMDPs with continuous and/or discrete states. We conclude the chapter by introducing Extended Algebraic Decision Diagrams, the data structure which is used to implement SDP.

In the next three chapters we provide novel exact solutions to subclasses of MDPs and POMDPs applied in various domains. In Chapter 5 we present a solution to the sequential market making problem by recasting it as a continuous state POMDP. We present a solution that provides sequentially optimal policies which maximise MM profit under a variety of inventory conditions and trader models, while simultaneously bounding loss.

In Chapter 6 we formulate a subclass of zero-sum continuous state stochastic games that can be solved exactly via parameterised linear optimisation. We calculate exact solutions to three empirical domains: (i) a continuous state generalisation of matching pennies; (ii) binary option valuation; and (iii) robust energy production.

In Chapter 7 we introduce Parameterised Hybrid MDPs (PHMDPs) with discrete and continuous states as a framework that allows for the first exact symbolic analysis of vaccination policies in an SIR epidemiological model [Kermack and McKendrick,

1927], as well exact solutions to the inverse learning of parameters in a multi-objective reward domain and sensitivity analyses of portfolio execution strategies.

Finally, in Chapter 8 we conclude by summarising the thesis and presenting some open problems for future work.

Frameworks for Sequential Decision Making

Sequential decision making is a *type* of decision making problem characterised by two unique features. The first defining characteristic of this problem type is the relationship between a decision made in the current instant and in future instances. The decision maker is presented with a sequence of decisions and needs to decide the current action by taking into account its effect on the solution to future decisions. The second characteristic is uncertainty in the consequences of each action. Uncertainty may be due to incomplete information about the environment, uncertain action effects or an ill-defined utility. In a financial market context, a trading agent acting with an electronic financial market may choose to retain a financial asset whose value it expects to appreciate in the future, rather than selling it for an immediate profit. The trading agent cannot be certain how the market value of the asset, or the market as a whole will move in the future, as a consequence of its action.

In this chapter we present fundamental frameworks that allow an agent to construct algorithms to make sequentially optimal decisions under progressively increasing levels of uncertainty. We begin with the Markov Decision Process (MDP) framework that captures uncertainty in the rewards or utility. We introduce the components of the MDP framework as well as solution concepts that are required to derive solutions to MDPs. This is followed by an illustration of how MDPs can be used to encapsulate a financial market domain. We then introduce Partially Observable Markov Decision Processes (POMDPs) which provides an extension to MDPs by incorporating uncertainty about the state of the environment. We introduce the components of the POMDP framework as well as solution concepts that are required to derive solutions to POMDPs. This is followed by an example of how POMDPs can be used to encapsulate a financial market domain. We conclude by briefly introducing Multi-Armed Bandits, which capture uncertainty in the rewards of selecting a particular action and Multi-Agent Frameworks which extend MDPs and POMDPs into settings with more than one agent.

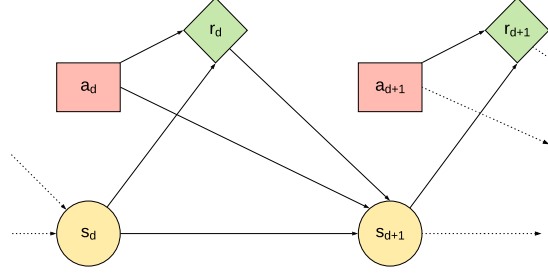


Figure 2.1: Graphical Model representation of an MDP. At every epoch $d \in \mathcal{D}$, action $a_d \in \mathcal{A}_d$ is executed in the current state s_d . This affects the transition to the next state s_{d+1} , while also obtaining the reward r_d .

2.1 Markov Decision Processes

Markov Decision Processes (MDPs) were first introduced as a way to model problems in the field of operations research [Bellman, 1957]. Since then it has become the de-facto standard for sequential decision making in fully-observable environments in the field of artificial intelligence. We begin this section with a formal description of the MDP framework followed by an exposition of the essential solution concepts required to work with and solve MDPs.

2.1.1 The Markov Decision Process Framework

A Markov Decision Process (MDP) can be defined in its most general form by the tuple $\langle \mathcal{D}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$. In the proceeding sections we describe each element of the tuple in turn. A graphical model representation of an MDP is given in Figure (2.1).

Decision epoch $\mathcal{D}$

The set of decision epochs $\mathcal{D}$ denote the points in time at which an agent must execute an action. $\mathcal{D}$ can be either discrete or continuous [Guo and Liu, 2001]; finite or infinite subsets of the real line. Decision problems with finite or infinite $\mathcal{D}$ are known as *finite* horizon and *infinite* horizon problems, respectively. We assume that decision epochs and horizons are finite in this thesis.

State space $\mathcal{S}$

The state space $\mathcal{S}$ contains the set of states in the environment. $\mathcal{S}$ may be finite [Derman, 1970], countably infinite [Kearns and Koller, 1999], uncountable [Boutilier et al., 1999] or a mixture of all three [Puterman, 1994, Feng et al., 2004], and may also depend on $d \in \mathcal{D}$. In this thesis we assume that the state space $\mathcal{S}$ is expressed in its most general form, $\mathcal{S} = \bigcup_{d \in \mathcal{D}} \mathcal{S}_d$.

In an MDP the state space is assumed to be fully observable to the agent.

Action space $\mathcal{A}$

The action space $\mathcal{A}$ contains the mechanisms by which an agent interacts with the environment. $\mathcal{A}$ can be finite, countably infinite or uncountable and may depend on the decision epoch $d \in \mathcal{D}$ and state $s \in \mathcal{S}$. In this thesis we assume that the action space $\mathcal{A}$ is expressed in its most general form $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \bigcup_{d \in \mathcal{D}} \mathcal{A}_{s,d}$, where $\mathcal{A}_{s,d}$ denotes the set of allowable actions in state $s \in \mathcal{S}$ and epoch $d \in \mathcal{D}$.

Transition function $\mathcal{T}$

The transition function captures the stochastic nature of action effects and is typically defined as a probability distribution $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{D} \rightarrow [0, 1]$. The transition function defines how the environment changes in response to an agent's action at decision epoch $d \in \mathcal{D}$.

An MDP imposes the first-order Markov and stationarity assumptions on the transition function $\mathcal{T}$, which stipulates that for $d, d' \in \mathcal{D}$, $a \in \mathcal{A}$ and $s, s' \in \mathcal{S}$, $\mathcal{T}(s, a, s', d) = \mathcal{T}(s, a, s', d')$. Henceforth, we drop the dependency on d and note that $\forall s, a, s', d \mathcal{T}(s, a, s', d) = \mathcal{T}(s, a, s')$.

Reward function $\mathcal{R}$

The reward function $\mathcal{R}$ encodes the preferences of the agent and is most generally defined by $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{D} \rightarrow \mathbb{R}$. The agent receives a reward from the environment in response to an action executed in a given state $s \in \mathcal{S}$ at epoch $d \in \mathcal{D}$.

An MDP imposes the first-order Markov and stationarity assumptions on the specification of $\mathcal{R}$, i.e. for $d, d' \in \mathcal{D}$, $a \in \mathcal{A}$ and $s, s' \in \mathcal{S}$, $\mathcal{R}(s, a, s', d) = \mathcal{R}(s, a, s', d')$. Henceforth, we drop the dependency on d and note that $\forall s, a, s', d \mathcal{R}(s, a, s', d) = \mathcal{R}(s, a, s')$.

2.1.2 Solution Concepts

The goal of an agent within an MDP is to calculate a mapping from states to actions, known as a policy, that maximises a predefined performance criterion over a specified decision epoch $\mathcal{D}$. In this section we introduce histories, policies, performance criterion and value functions, concepts that underlie all MDP solution techniques. We present the aforementioned concepts for discrete finite $\mathcal{S}$ and $\mathcal{A}$ spaces.

History

An execution history of an MDP is a sequence of pairs of states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$ that an agent has visited and actions executed in those states up until a particular decision epoch $d \in \mathcal{D}$. A history can be specified as:

$$\mathcal{H}_d = ((s_1, a_1), (s_2, a_2), \dots, (s_{d-1}, a_{d-1}), s_d), \quad (2.1)$$

where $d \in \mathcal{D}$, $s_d \in \mathcal{S}$ and $a_d \in \mathcal{A}$. The set of all possible execution histories is given by $\mathcal{H} = \bigcup_{d \in \mathcal{D}} \mathcal{H}_d$.

Policies

A policy specifies the actions $a_d \in \mathcal{A}_d$ that an agent executes at epoch $d \in \mathcal{D}$ to control the evolution of the environment. There are four main families of policies, as shown in Table (2.1).

Table 2.1: MDP Policy Families.

	Stochastic	Deterministic
History-dependent	$\pi : \mathcal{D} \times \mathcal{H} \rightarrow \mathcal{P}(\mathcal{A})$	$\pi : \mathcal{D} \times \mathcal{H} \rightarrow \mathcal{A}$
Markov	$\pi : \mathcal{D} \times \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$	$\pi : \mathcal{D} \times \mathcal{S} \rightarrow \mathcal{A}$

A policy, in its most general form, can be defined as $\pi : \mathcal{D} \times \mathcal{H} \rightarrow \mathcal{P}(\mathcal{A})$, where $\mathcal{D}$ is the decision epoch, $\mathcal{H}$ is the set of all execution histories of the MDP and $\mathcal{P}$ is a probability distribution on $\mathcal{A}$. Policies can be classified based on their dependence on past information and the nature of $\mathcal{P}(\mathcal{A})$. The most restrictive definition of a policy, in the scheme presented in Table (2.1), is the deterministic Markovian policy, which for a given $d \in \mathcal{D}$, considers the current state $s \in \mathcal{S}$ and returns a single action $a \in \mathcal{A}$. A policy is termed *stationary* if $\forall d, \pi_d = \pi$, and non-stationary, otherwise. In many infinite horizon settings, a stationary policy is optimal and hence the dependence on $\mathcal{D}$ can be dropped. Together, the stationary, deterministic, Markovian and stationary, stochastic, Markovian families contain *optimal* policies for the most common performance criteria.

Performance Criterion

Solving an MDP involves searching for a policy, within the policy families discussed previously, which optimises a specified performance criterion. As a result of implementing a policy the agent implicitly induces a probability distribution on the set of all possible reward sequences. One popular performance criterion used in MDPs are presented in Table (2.2).

Table 2.2: MDP Performance Criteria.

Finite	Infinite
$\mathbb{E}^\pi \left[\sum_{d=1}^{ \mathcal{D} } \gamma^d \cdot R_d s_0 \right]$	$\mathbb{E}^\pi \left[\sum_{d=1}^{\infty} \gamma^d \cdot R_d s_0 \right]$

The finite horizon criterion asserts that the decision making agent has $|\mathcal{D}|$ epochs to interact with the environment. The infinite horizon criterion is used when there is an infinite number of decision epochs. The discount factor γ is used to encode an agents' preference for the realisation of a reward. In the finite horizon case $\gamma \in [0, 1]$, whereas in the infinite horizon case $\gamma \in [0, 1)$. Setting $\gamma = 0$ encodes a myopic agent that only considers immediate rewards, whereas setting $\gamma \rightarrow 1$ reflects an indifference of the agent to when a particular reward is realised. The discount factor γ also ensures that the sum of rewards is finite.

The two performance criteria in Table (2.2) share two common features: (i) they are all additive in $\mathcal{R}$; and (ii) they are defined with respect to an expectation operation $\mathbb{E}[\cdot]$. The expectation operator is used to account for the distribution of reward sequences for a given policy π and initial state s_0 . Furthermore, the expected cumulative sum of rewards leads to the Bellman optimality principle [Bellman, 1957], which underlies all dynamic programming based solution techniques.

Policies can be compared on the basis of expected utility, that is, the preference for all different realisations of the reward sequence obtained by following the policy and the probability of each occurring.

Value Functions

A value function links any state $s \in \mathcal{S}$ to the utility gained by an agent following a policy π under a specified performance criterion. The value of a state $s \in \mathcal{S}$ under a given policy π , $V^\pi(s)$, is the expected reward when starting in s and following π thereafter. A value function under the finite performance criterion is given by:

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_d^{|D|} \gamma^d \cdot R_d | s_0 = s \right]. \quad (2.2)$$

Equation (2.2) shows that the expected value of a state s is defined over the state-action distribution starting in state s and following policy π .

The search for an optimal policy π^* can be framed in terms of value functions. In fact, value functions define a partial ordering over policies. That is, $\pi \geq \pi'$ if, and only if, $V^\pi(s) \geq V^{\pi'}(s) \forall s \in \mathcal{S}$. The *optimal* value function V^{π^*} , under the optimal policy π^* , can be defined as:

$$\forall s \in \mathcal{S} \quad V^{\pi^*}(s) \geq V^\pi(s).$$

We note that an optimal policy π^* may not be unique, but all optimal policies have the same optimal value function V^{π^*} .

The state-action value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ can be defined as the expected return starting from state s , taking action a and following policy π , thereafter. The state-action value function under the finite performance criterion is given by:

$$Q^\pi(s, a) = \mathbb{E}^\pi \left[\sum_d^{|D|} \gamma^d \cdot R_d | s_d = s, a_d = a \right]. \quad (2.3)$$

The value function presented in Equation (2.2) satisfies a particular recursive relationship. For any policy π and state s , the following consistency condition holds

between the value of s and the value of its possible successor states:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}^\pi \left[\sum_d^{|\mathcal{D}|} \gamma^d \cdot R_d | s_0 = s \right] \\ &= \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') [\mathcal{R}(s, \pi(s), s') + \gamma \cdot V^\pi(s')]. \end{aligned} \quad (2.4)$$

Equation (2.4) is known as the Bellman equation for V^π . It shows that the expected value of a state s is defined in terms of the immediate reward and values of possible next states acting under policy π weighted by their transition probabilities and a discount factor. The value function V^π is the unique solution to its Bellman equation.

2.1.3 MDPs for Financial Markets

The Markov Decision Process framework presented thus far is, despite its simplicity, a powerful framework that is able to encapsulate many financial market formulations. In this section we demonstrate how a financial market trading agent can be expressed as an MDP.

Suppose a trading agent holds one unit of a financial derivative $\mathcal{F}$ with an underlying asset $\mathcal{U}$ and maturity $|\mathcal{D}|$. At each time period $d < |\mathcal{D}|$, the agent can either sell $\mathcal{F}$ at the current spot price $\mathcal{F}_d$ or hold on to it. When $d = |\mathcal{D}|$ the derivative will be sold, if it hasn't previously, at the price $\mathcal{F}_{|\mathcal{D}|} = g(\mathcal{U}_{|\mathcal{D}|})$, where $g(\mathcal{U}_{|\mathcal{D}|})$ is the *payoff* function of the derivative $\mathcal{F}$. A trading episode ends as soon as $\mathcal{F}$ is sold.

We now proceed to formalise the trading agent using the MDP framework presented in Section 2.1.1. $\mathcal{S} = \langle m, i \rangle$, where $m \in \mathbb{R}^+$ represents the value of the derivative and $i \in \mathbb{N}$ represents the trading agents' inventory. $\mathcal{A} = \langle \text{sell}, \text{hold} \rangle$, where *sell* connotes selling the derivative and *hold*, otherwise. In this instance $\mathcal{A}$ is discrete and finite. If we assume that the trading agents' actions only have an effect on its inventory and not the market value of the derivative, then we can write the transition function $\mathcal{T}$ for each state variable in $\mathcal{S}$ as:

$$\begin{aligned} \mathcal{T}(i_{d+1} = i_d - 1 | i_d = i, a_d = \text{sell}, m_d) &= \begin{cases} 1.0 & \text{if } i \geq 1 \\ 0.0 & \text{otherwise} \end{cases} \\ \mathcal{T}(i_{d+1} = i_d | i_d = i, a_d = \text{sell}, m_d) &= 0.0 \\ \mathcal{T}(i_{d+1} = i_d + 1 | i_d = i, a_d = \text{sell}, m_d) &= 0.0 \\ \mathcal{T}(i_{d+1} = i_d - 1 | i_d = i, a_d = \text{hold}, m_d) &= 0.0 \\ \mathcal{T}(i_{d+1} = i_d | i_d = i, a_d = \text{hold}, m_d) &= 1.0 \\ \mathcal{T}(i_{d+1} = i_d + 1 | i_d = i, a_d = \text{hold}, m_d) &= 0.0 \\ \mathcal{T}(m_{d+1} = m_d | i_d = i, a_d = a, m_d) &= 1.0 \end{aligned}$$

The reward function specifies the return to the trading agent contingent upon its

actions. $\mathcal{R}$ can be specified as follows:

$$\mathcal{R}(m, i, a) = \begin{cases} m_d & \text{if } a = \text{sell} \text{ and } d < |\mathcal{D}| \\ g(\mathcal{U}_{|\mathcal{D}|}) & \text{if } d = |\mathcal{D}| \text{ and } i \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

We note that even though the transition function $\mathcal{T}$ is defined as stochastic, for the purposes of this thesis, they are in fact deterministic. In Section 5.2 we formalise deterministic transition functions in a stochastic framework using delta functions.

The MDP trading agent formalised above can now be solved using the algorithms presented in Chapter 3.

2.2 Partially Observable Markov Decision Processes

Partially Observable Markov Decision Processes (POMDPs) allow for optimal decision making in environments that are only *partially* observable to the agent [Kaelbling et al., 1998]. The agent observes the process but does not know its state. In general, partial observability stems from two sources: noisy sensor readings or multiple states giving the same sensor reading. The POMDP captures partial observability in a probabilistic observation model, which relates possible observations to states.

We begin this section with a formal description of the POMDP framework followed by an exposition of the essential solution concepts required to work with and solve POMDPs.

2.2.1 The POMDP Framework

A Partially Observable Markov Decision Process (POMDP) can be defined in its most general form by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, \mathcal{Z}, \mathcal{D} \rangle$. A POMDP is comprised of an underlying MDP, extended with an observation space and observation function. In the proceeding sections we describe each element of the tuple that is unique to POMDPs in turn. A graphical model representation of a POMDP is given in Figure (2.2).

Observation space $\mathcal{O}$

The observation space $\mathcal{O}$ encapsulates the set of observations that an agent can perceive from the environment. The number of observations may be finite, countably infinite or continuous, and may depend on $d \in \mathcal{D}$, in which case $\mathcal{O} = \bigcup_{d \in \mathcal{D}} \mathcal{O}_d$. The observation space $\mathcal{O}$ may differ from $\mathcal{S}$.

Observation function $\mathcal{Z}$

The observation function $\mathcal{Z} : \mathcal{O} \times \mathcal{A} \times \mathcal{S} \times \mathcal{D} \rightarrow [0, 1]$ links the observations $o \in \mathcal{O}$ to the states $s \in \mathcal{S}$ by specifying the probability of perceiving an o after executing an action $a \in \mathcal{A}$ in an underlying state s . Observations provide partial information to

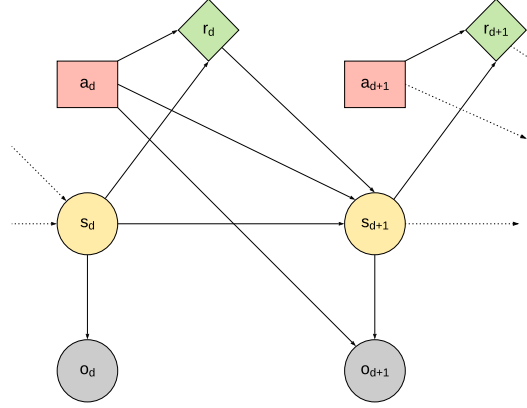


Figure 2.2: Graphical Model representation of a POMDP. At every epoch $d \in \mathcal{D}$, the agent perceives observation $o_d \in \mathcal{O}_d$ but does not know the underlying state $s_d \in \mathcal{S}_d$. Action $a_d \in \mathcal{A}_d$ is executed in the current state s_d , which affects the transition to the next state s_{d+1} while also obtaining the reward r_d .

the agent about the underlying state, since the same observation may arise from two different states.

A POMDP imposes the first-order Markov and stationarity assumptions on the observation function $\mathcal{Z}$, which stipulates that for $d, d' \in \mathcal{D}$, $a \in \mathcal{A}$, $o \in \mathcal{O}$ and $s, s' \in \mathcal{S}$, $\mathcal{Z}(o, a, s, d) = \mathcal{Z}(o, a, s, d')$. Henceforth, we drop the dependency on d and note that $\forall o, a, s', d \mathcal{Z}(o, a, s') = \mathcal{Z}(o, a, s', d')$.

2.2.2 Solution Concepts

The goal of an agent within a POMDP is to calculate a mapping from observations to actions, known as a policy, that maximises a predefined performance criterion over a specified horizon. In this section we introduce belief states, histories, policies, value functions and α -vectors, concepts that underlie all POMDP solution techniques. We present the aforementioned concepts for discrete finite $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$ spaces, unless stated otherwise.

History

An observation history of a POMDP is a sequence of pairs of observations $o \in \mathcal{O}$ and actions $a \in \mathcal{A}$ that an agent has perceived from the environment and actions executed according to the agent's belief b_d up until a particular decision epoch $d \in \mathcal{D}$. A POMDP history can be specified as:

$$\mathcal{H}_d = ((o_1, a_1), (o_2, a_2), \dots, (o_{d-1}, a_{d-1}), o_d), \quad (2.5)$$

where $d \in \mathcal{D}$, $o_d \in \mathcal{O}$ and $a_d \in \mathcal{A}$. The set of all possible observation histories is given by $\mathcal{H} = \bigcup_{d \in \mathcal{D}} \mathcal{H}_d$.

Belief State

A POMDP agent must choose its actions without complete knowledge about the state of the environment. As a result, it is not possible to define value functions and policies for POMDPs in the manner that was used for MDPs.

A POMDP agent must instead rely on the available information that it has received from the environment thus far. The simplest way to elicit and encode this information is to rely on the observation history $\mathcal{H}$ of the process since epoch $d = 0$ and an initial distribution over states $\mathcal{S}$. Together this information defines a controlled Markov process. The main drawback of this approach is that the amount of information in the observation history grows at each epoch, which quickly becomes intractable.

A more efficient representation for the observation history and initial distribution over $\mathcal{S}$ is to utilise sufficient statistics relative to the process [Bertsekas, 2000]. These sufficient statistics obey the Markov property and preserve enough information to allow an agent to control the underlying process. The main advantage of using sufficient statistics is their more compact representation; the size of sufficient statistics do not grow over time. The most widely used sufficient statistic for POMDPs are *belief states*.

A belief state is a $b_d \in \mathcal{P}(\mathcal{S})$, a probability distribution over state $\mathcal{S}$ for $d \in \mathcal{D}$. The set of belief states $\mathcal{B}$ is defined as the space of all probability distributions over $\mathcal{S}$. An agent assumes some initial belief $b_0 \in \mathcal{P}(\mathcal{S})$ about the state of the environment and updates its belief for each state after executing an action $a \in \mathcal{A}$ and observing $o \in \mathcal{O}$ according to the following rule:

$$\begin{aligned}
 b_d(s') &= \mathbb{P}(s'|o, a, b_{d-1}) \\
 &= \frac{\mathbb{P}(o|s', a, b_{d-1})\mathbb{P}(s'|a, b_{d-1})}{\mathbb{P}(o|a, b_{d-1})} \\
 &= \frac{\mathbb{P}(o|s', a) \sum_{s \in \mathcal{S}} \mathbb{P}(s'|s, a, b_{d-1})\mathbb{P}(s|a, b_{d-1})}{\mathbb{P}(o|a, b_{d-1})} \\
 &= \frac{\mathbb{P}(o|s', a) \sum_{s \in \mathcal{S}} \mathbb{P}(s'|s, a) b_{d-1}(s)}{\mathbb{P}(o|a, b_{d-1})} \\
 &= \frac{\mathcal{Z}(o, a, s') \sum_{s \in \mathcal{S}} \mathcal{T}(s, a, s') b_{d-1}(s)}{\mathbb{P}(o|a, b_{d-1})}
 \end{aligned}$$

The current belief $b_d(s')$ is a sufficient statistic for the observation history $\mathcal{H}_d$.

Performance Criterion

Solving a POMDP involves searching for a policy, within the policy families discussed previously, which optimises a specified performance criterion. The family of performance criteria is similar to that used within MDPs in Section 2.1.2.

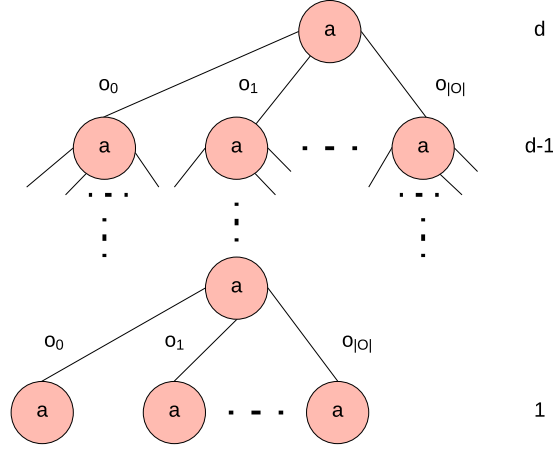


Figure 2.3: A policy tree for epoch d . When in a node, the associated action is applied and then, according to the observation received, the next node is reached by descending the branch labelled by the observation.

Policy Trees and Conditional Plans

In the context of POMDP's an agent must decide its action based on its history of actions and observations. Hence, a POMDP policy assumes a more complicated form than an MDP policy.

A POMDP policy, in its most general form, can be defined as $\pi : \mathcal{D} \times \mathcal{H} \rightarrow \mathcal{P}(\mathcal{A})$, where $\mathcal{D}$ is the decision epoch, $\mathcal{H}$ is the set of all execution histories of the POMDP and $\mathcal{P}$ is a probability distribution on $\mathcal{A}$. In the proceeding sections we introduce a possible representation using $\mathcal{H}$.

Policy Trees A policy tree can be constructed via the process of backward induction. At the penultimate decision epoch, the agent can only execute one action; with two epochs left, the agent can execute an action, receive an observation and then execute its final action. Hence, in general, an agent's d -step policy can be represented as a *policy tree* with depth d .

An example of a policy tree is shown in Figure (2.3). Each node prescribes an action $a \in \mathcal{A}$ at a particular $d \in \mathcal{D}$; then, depending on the resulting observation $o \in \mathcal{O}$ a branch is followed to a node on the next level, which determines the next action. There are $|\mathcal{A}|^{\frac{|\mathcal{O}|^{\mathcal{D}} - 1}{|\mathcal{O}| - 1}}$ many possible $\mathcal{D}$ length policy trees.

Policy trees are sufficient for policies with a finite $\mathcal{D}$ but are often inappropriate for infinite $\mathcal{D}$ policies. When $\mathcal{D}$ is infinite and policies are cyclic, Finite State Controllers are a possible solution [Kaelbling et al., 1998].

Conditional Plans A policy tree can also be defined recursively in the form of a *conditional plan*. A conditional plan β specifies the action to execute and what to do contingent on an observation received. The set of all conditional plans available to

an agent at decision epoch $d \in \mathcal{D}$ can be defined as:

$$\Gamma_d = \{(a, \nu) | a \in \mathcal{A}, \nu : \mathcal{O} \rightarrow \Gamma_{d-1}\}, \quad (2.6)$$

where $\nu : \mathcal{O} \rightarrow \Gamma_{d-1}$ is a stage-dependent observation strategy [Braziunas, 2003]. A conditional plan of epoch d is defined recursively using the conditional plans of epoch $d - 1$, hence it corresponds to a policy tree of depth d . At the last step, a conditional plan returns an action.

Value Functions

The value function of a policy represented by a d -step conditional plan $\beta = \langle a, \nu \rangle$ can be written recursively as:

$$V_\beta^d(s) = \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) \left[\mathcal{R}(s, a, s') + \gamma \cdot \sum_{o \in \mathcal{O}} \mathbb{P}(o|s', a) V_{\nu(o)}^{d-1}(s') \right]. \quad (2.7)$$

Since the state $s \in \mathcal{S}$ is not fully known, the value function in Equation (2.7) must be written with respect to a belief $b \in \mathcal{P}(\mathcal{S})$. We can write the value function with respect to a belief as the expectation of executing the conditional plan β at each $s \in \mathcal{S}$:

$$V_\beta^d(b) = \sum_{s \in \mathcal{S}} b(s) V_\beta^d(s). \quad (2.8)$$

An optimal policy π^* is one which specifies for each belief state b an optimal action. The *optimal* value function under π^* is defined by:

$$V^{\pi^*}(b) = \max_{\beta \in \Gamma} \sum_{s \in \mathcal{S}} b(s) V_\beta(s), \quad (2.9)$$

which simply enumerates all the possible policy trees in the set Γ .

The value function in Equation (2.9) is Piecewise Linear and Convex (PWLC) and parameterised by a finite set of vectors [Sondik, 1971]. Since the value function is defined on the belief space $\mathcal{P}(\mathcal{S})$, which is $|\mathcal{S}| - 1$ dimensional, each linear segment used to represent it is also $|\mathcal{S}|$ -dimensional. Hence, value functions are linear with respect to the belief b . Furthermore, the optimal value is the upper surface (i.e. maximum) of the collection of value functions of policies in Γ , thus maintaining a convex region.

α -Vectors and α -Functions

Equation (2.9) can be redefined using a finite set of α -vectors. An α -vector is an $|\mathcal{S}|$ -dimensional vector of values for a conditional plan β over all states $s \in \mathcal{S}$. The value function at a given belief $b \in \mathcal{P}(\mathcal{S})$ can be written as:

$$V_d(b) = \max_{\alpha \in \Gamma_d} \sum_{s \in \mathcal{S}} \alpha(s) \cdot b(s), \quad (2.10)$$

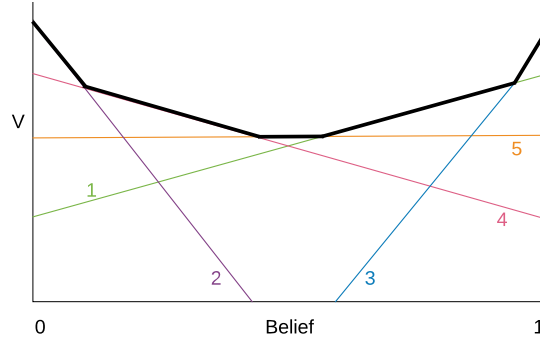


Figure 2.4: Five α -vectors representing the Value function of a 2 state POMDP. The belief space is on the x-axis and the value is on the y-axis. The linear segments that constitute the value function are in bold.

where Γ_d contains all α -vectors at epoch d . Each of the vectors $\alpha_i \in \Gamma_d$ can be written as:

$$\alpha_i(b, s) = \mathcal{R}(s, a) + \gamma \sum_{o \in \mathcal{O}} \sum_{s' \in \mathcal{S}} \mathbb{P}(s', o | s, a) \cdot \alpha_{i-1}(b, s).$$

A maximum of $|\mathcal{A}| |\Gamma_{d-1}|^{|\mathcal{O}|}$ vectors are needed to represent V_d . In practice, the number of α -vectors used to represent V_d can be reduced by removing *dominated* vectors. A dominated vector is one that does not influence the value function. As a result, they can be pruned from the value function representation.

The convexity of value functions implies that the value of a belief close to one of the corners of the belief simplex $\mathcal{P}(\mathcal{S})$ will be high. Each α -vector is generated for a particular $a \in \mathcal{A}$. The optimal action at epoch d is given by Equation (2.10). Thus, Γ_d encodes not only the value, but also the optimal policy. Using this formulation, solution methods typically focus on the computation of the α -vectors.

Figure (2.4) shows an example of five α -vectors, $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$, representing the value function of a 2 state POMDP. The upper surface of α -vectors represents the optimal value function with a PWLC property.

The results of Sondik [Sondik, 1971] can be generalised to continuous state POMDPs, while still assuming discrete $\mathcal{A}$ and $\mathcal{O}$. In this setting, the value function can be represented as a set of α -functions. The value function in a continuous state POMDP at a given belief $b \in \mathcal{P}(\mathcal{S})$ can be calculated using:

$$V_d(b) = \sup_{\alpha \in \Gamma_d} \int \alpha(s) \cdot b(s) d\mathcal{S}, \quad (2.11)$$

for an appropriate continuous set of α -functions [Porta et al., 2006]. Furthermore, for continuous $\mathcal{S}$, discrete $\mathcal{A}$ and $\mathcal{O}$, the finite horizon value function is PWLC.

2.2.3 POMDPs for Financial Markets

In this section we demonstrate how the POMDP framework can be used to specify an extension of the financial market trading agent presented in Section 2.1.3. A POMDP enables a trading agent to incorporate, amongst a plethora of other things, uncertainty about the *true* underlying value of the derivative $\mathcal{F}$.

$\mathcal{S} = \langle m, i \rangle$, where $m \in \mathbb{R}^+$ represents the true market value of the derivative and $i \in \mathbb{N}$ represents the trading agents' inventory. $\mathcal{A} = \langle \text{sell}, \text{hold} \rangle$, where *sell* connotes selling the derivative and *hold*, otherwise. $\mathcal{O} = \langle \text{higher}, \text{lower} \rangle$, where *higher* denotes the market price of the derivative increased since the last decision epoch $d < |\mathcal{D}|$ and *lower* denotes that the market price decreased since the last epoch. If, as in Section 2.1.3, we assume that the trading agents' transaction volume has no effect on the market price of the derivative, we can proceed to write the transition function $\mathcal{T}$ for each state variable in $\mathcal{S}$ as:

$$\begin{aligned} \mathcal{T}(i_{d+1} = i_d - 1 | i_d = i, a_d = \text{sell}, m_d) &= \begin{cases} 1.0 & \text{if } i \geq 1 \\ 0.0 & \text{otherwise} \end{cases} \\ \mathcal{T}(i_{d+1} = i_d | i_d = i, a_d = \text{sell}, m_d) &= 0.0 \\ \mathcal{T}(i_{d+1} = i_d + 1 | i_d = i, a_d = \text{sell}, m_d) &= 0.0 \\ \mathcal{T}(i_{d+1} = i_d - 1 | i_d = i, a_d = \text{hold}, m_d) &= 0.0 \\ \mathcal{T}(i_{d+1} = i_d | i_d = i, a_d = \text{hold}, m_d) &= 1.0 \\ \mathcal{T}(i_{d+1} = i_d + 1 | i_d = i, a_d = \text{hold}, m_d) &= 0.0 \\ \mathcal{T}(m_{d+1} = m_d | i_d = i, a_d = a, m_d) &= 1.0 \end{aligned}$$

The observation function specifies the relation between the state and the observation received. In this particular instance, the $\mathcal{Z}$ can be specified as follows:

$$\begin{aligned} \mathcal{Z}(o_{d+1} = \text{higher} | i_d = i, a_d = a, m_d) &= \begin{cases} 1.0 & \text{if } m_{d+1} \geq m_d \\ 0.0 & \text{otherwise} \end{cases} \\ \mathcal{Z}(o_{d+1} = \text{lower} | i_d = i, a_d = a, m_d) &= \begin{cases} 1.0 & \text{if } m_{d+1} < m_d \\ 0.0 & \text{otherwise} \end{cases} \end{aligned}$$

The reward function specifies the return to the trading agent contingent upon its actions. $\mathcal{R}$ can be specified as follows:

$$\mathcal{R}(m, i, a) = \begin{cases} m_d & \text{if } a = \text{sell} \text{ and } d < |\mathcal{D}| \\ g(\mathcal{U}_{|\mathcal{D}|}) & \text{if } d = |\mathcal{D}| \text{ and } i \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

The POMDP trading agent formulation can now be solved using the algorithms

presented in Chapter 3.

2.3 Other Decision Making Frameworks

MDPs and POMDPs are both frameworks that have been used to formalise sequential decision making under uncertainty. An MDP assumes that the agent has complete knowledge of the state of the environment whereas the POMDP only assumes partial observability of the environment. Despite these differences, both frameworks allow an agent to construct an optimal policy that maximises a specified performance criterion. MDPs and POMDPs are the two frameworks that will be the focus of this thesis and its novel contributions.

In this section we introduce alternate frameworks for sequential decision making. We begin by discussing the Multi-Armed Bandit, which despite its simplicity, addresses a fundamental dilemma of decision making known as the exploration-exploitation trade-off. The dilemma stems from the fact that an agent that only explores new possibilities will pay the cost of not being able to use the information gained, whereas an agent that only exploits existing knowledge may be incapable of adapting to changes in the environment and/or may be trapped with a sub-optimal decision. We conclude this section by offering alternate frameworks that address the single agent limitation of MDPs and POMDPs. These frameworks extend MDPs and POMDPs into *non-cooperative* and *cooperative* multi-agent systems whereby the interactions of several agents within the environment are modelled together.

2.3.1 Multi-Armed Bandits

Multi-Armed Bandits (MABs) are a simple mathematical framework for sequential decision making in uncertain environments that encapsulates the exploration-exploitation trade-off. The MAB framework was initially introduced in the context of clinical trials [Thompson, 1933, Robbins, 1952], where an agent must decide which treatment to use on a class of individuals, and has recently found traction in advertisement in e-commerce, news article recommendation [Mohan, 2010] and algorithmic market-making [Abernethy and Kale, 2013]. The utility of MABs is due to the expressiveness of the framework and the efficacy of their associated algorithms.

A Multi-Armed bandit can be defined in its most general form by the tuple $\langle \mathcal{A}, \mathcal{R}, \mathcal{D} \rangle$. The set of decision epochs $\mathcal{D}$ denote the points in time at which an agent must execute an action. $\mathcal{D}$ can be either discrete or continuous [Kaspi and Mandelbaum, 1998], finite or infinite subsets of the real line. The set $\mathcal{A}$ denotes a measurable space of actions and may consist of finite or countably infinite sets [Thompson, 1933, Robbins, 1952] or uncountable sets [Agrawal, 1995, Kleinberg, 2004]. Finally, $\mathcal{R}$ describes the *distribution* of rewards associated with each action in $\mathcal{A}$. An agent observes the reward associated with only the selected action, the rewards associated with the other actions are not revealed. The reward process of an MAB may be stochastic [Lai and Robbins, 1985], adversarial [Auer et al., 2003] or Markovian [Gittins, 1979].

The goal of an MAB solution is to specify a strategy for action selection that optimises a reward criterion. In MABs with stochastic rewards and full-information, the merits of an action selection strategy are compared to that of an *optimal* strategy that, for each time step, selects the best action. The cumulative regret [Lai and Robbins, 1985] specifies the difference between the agent’s actual accumulated reward to the amount they could have accumulated had they consistently chosen the optimal action:

$$\bar{R}_{|\mathcal{D}|} = |\mathcal{D}| \cdot \mu^* - \mathbb{E} \left[\sum_d^{|\mathcal{D}|} \mu_a(d) \right], \quad (2.12)$$

where μ_a is the mean of the probability distribution associated with action $a \in \mathcal{A}$ and $\mu^* = \max_{a \in \mathcal{A}} \mu_a$ is the expected reward from the best action. A good action selection strategy aims to minimise the cumulative regret.

Multi-Armed Bandits have been utilised by researchers from a variety of disciplines, including economics, psychology and computer science. More recently, the MAB framework has been used to calculate *approximate* solutions to MDPs and POMDPs with large $\mathcal{S}$, $\mathcal{A}$ or $\mathcal{O}$ spaces via Monte Carlo Tree Search (MCTS), which we discuss in Chapter 3. While this thesis focuses on symbolic solutions to MDPs and POMDPs, we discuss potential connections with MABs and MCTS in Chapter 8.

2.3.2 Multi-Agent Frameworks

Stochastic Games (SGs), also known as Markov Games [Shapley, 1953], can be framed as an extension of MDPs to the multi-agent setting. An SG is defined by the tuple $\langle n, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$. $n \in \mathbb{Z}^+$ specifies the number of agents within the environment. Actions $\mathcal{A} = \mathcal{A}_1 \times \dots \times \mathcal{A}_n$, denotes the set of joint actions where $\mathcal{A}_i$ is the set of actions available to agent $i \in n$. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{D} \rightarrow [0, 1]$ encapsulates how the environment changes in response to the *joint* action $a \in \mathcal{A}$. The reward function is defined as $\mathcal{R} = \mathcal{R}_1 \times \dots \times \mathcal{R}_n$, where $\mathcal{R}_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function of agent $i \in n$. Finally, $\mathcal{S}$ is defined in the same way as in the MDP framework. An SG can be instantiated as an MDP by setting $n = 1$.

Littman [Littman, 1994] utilised the SG framework to introduce Minimax-Q, an algorithm which extends traditional MDP Q-learning to two agent environments where each agent has diametrically opposed goals and discrete $\mathcal{S}$.

Multi-agent Markov Decision Processes (MMDPs) [Boutilier, 1999] extend MDPs to cooperative multi-agent settings. MMDPs utilise a reward function that is common to all agents, with one unified goal. Stochastic games [Shapley, 1953], on the other hand, consider separate reward functions for each agent, with possibly distinct goals.

An MMDP can be defined in its most general form by the tuple $\langle n, \mathcal{D}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$. $n \in \mathbb{Z}^+$ specifies the number of agents within the environment. Actions $\mathcal{A} = \mathcal{A}_1 \times \dots \times \mathcal{A}_n$, denotes the set of joint actions where $\mathcal{A}_i$ is the set of actions available to agent $i \in n$. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{D} \rightarrow [0, 1]$ encapsulates how the environment changes in response to the *joint* action $a \in \mathcal{A}$. The reward function has a similar dependence on the joint action $a \in \mathcal{A}$ and is defined as $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{D} \rightarrow$

$\mathbb{R}$. Finally, $\mathcal{S}$ and $\mathcal{D}$ are defined in the same way as in the MDP framework. An MMDP can be framed as a large MDP by taking the joint action space to be the set of basic actions.

Solving an MMDP consists of finding a *joint* optimal policy $\pi = \langle \pi_1, \dots, \pi_n \rangle$, where π_i is the policy of agent $i \in n$. The individual policy of agent i is defined as $\pi_i : \mathcal{S} \rightarrow \mathcal{A}_i$. Since an MMDP can be viewed as an MDP, value functions and other solution concepts are applicable in the MMDP setting. However, MMDPs require that each agent has complete knowledge of the state of the environment.

In general, agents may not always be able to observe the true underlying state of the environment or other agents. The Decentralised Markov Decision Process (DEC-MDP) and Decentralised Partially Observable Markov Decision Process (DEC-POMDP) [Bernstein et al., 2000] frameworks specify that agents must base their decision on a partial view of the environment.

A DEC-POMDP can be defined by the tuple $\langle n, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, \mathcal{Z}, \mathcal{D} \rangle$. The observation space $\mathcal{O} = \mathcal{O}_1 \times \dots \times \mathcal{O}_n$, where $\mathcal{O}_i$ denotes the set of joint observations perceivable by agent $i \in n$. The observation function $\mathcal{Z} : \mathcal{O} \times \mathcal{A} \times \mathcal{S} \times \mathcal{D} \rightarrow [0, 1]$ specifies the probability that each agent $i \in n$ observes o_i when the agents execute the *joint* action a . The other components of the tuple are as previously defined for MMDPs. A DEC-POMDP reduces to a DEC-MDP if the state of the environment is joint fully observable, that is, the current state is entirely determined by the amalgamation of all agents' observations.

Solving a DEC-POMDP or a DEC-MDP entails calculating its joint optimal policy $\pi = \langle \pi_1, \dots, \pi_n \rangle$, where π_i is the policy of agent $i \in n$. The individual policy of agent i can be defined as a mapping from an agents' belief states to an action $a_i \in \mathcal{A}$, as in the POMDP setting.

2.4 Summary

Sequential decision making is a fundamental concern for agents acting in uncertain environments. Despite its necessity, the decision making process is complicated by the need to consider the effect of a decision on the solution to future decisions, as well as uncertainty about its consequences in the current epoch. In this chapter we presented two main frameworks for sequential decision making, the Markov Decision Process and the Partially Observable Markov Decision Process. We also briefly introduced Multi-Armed Bandits and Multi-Agent frameworks for sequential decision making.

Multi-Armed Bandits, the simplest of all of the frameworks, distils the decision making problem into one of action selection under uncertain rewards. Markov Decision Processes generalise Multi-Armed Bandits by incorporating the notion of state space $\mathcal{S}$, which allow for complex representations of in the environment and uncertainty in the effects of an action via the transition function $\mathcal{T}$. The Markov Decision Process formulations considered in this thesis assume that each component of $\langle \mathcal{D}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ is stationary, that is, they do not change over time. Furthermore, we assume that $\mathcal{T}$ and $\mathcal{R}$ are Markovian, that is, they only depend on the current state

and action. We also assume that $\mathcal{A}$ is discrete, $\mathcal{D}$ is infinite and $\mathcal{S}$ is continuous, unless stated otherwise.

Partially Observable Markov Decision Processes were presented as the most general framework, extending Markov Decision Processes by adding uncertainty to the observation of state. Partial observability of the state space is specified through the observation space $\mathcal{O}$ and observation function $\mathcal{Z}$, which explicitly link observations to states in $\mathcal{S}$. In this thesis we assume that each component of $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, \mathcal{Z}, \mathcal{D} \rangle$ is stationary. Furthermore, we assume that $\mathcal{T}$, $\mathcal{R}$ and $\mathcal{Z}$ are Markovian and $\mathcal{O}$ is discrete. We also assume that $\mathcal{A}$ is discrete and $\mathcal{D}$ is infinite and $\mathcal{S}$ is continuous, unless stated otherwise.

The MDP and POMDP frameworks are limited by their single agent view of the environment. Stochastic Games, Multi-agent Markov Decision Processes, Decentralised Markov Decision Process and Decentralised Partially Observable Markov Decision Process were presented as multi-agent extensions of the aforementioned single agent settings.

In the forthcoming chapters we demonstrate how MDPs and POMDPs can be solved. We begin by exploring exact discrete solution techniques for these frameworks that calculate optimal policies for all states in $\mathcal{S}$.

Exact Discrete Solutions

Markov Decision Processes and Partially Observable Markov Decision Processes are two expressive frameworks capable of encapsulating a multitude of sequential decision making problems. In Chapter 2 we introduced the components of each framework and the solution concepts required to solve them. The ultimate goal of both frameworks is to calculate an optimal policy π^* , which ascribes an action to be executed in a particular state, or belief state, of the environment. In this chapter we introduce algorithms that provide an *exact* solution to discrete formulations of MDPs and POMDPs.

Exact solution techniques aim to calculate a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, where $\mathcal{S}$ is the *complete* state space. As such, the feasibility of these techniques are highly dependent upon the particular instantiation of the sequential decision making framework [Kearns et al., 2002]. In this Chapter we limit our discussion to algorithms that calculate policies for discrete and continuous $\mathcal{S}$, discrete $\mathcal{A}$, discrete $\mathcal{O}$ and finite $\mathcal{D}$ frameworks.

We begin our discussion with techniques for discrete $\mathcal{S}$ MDPs where we introduce Policy Iteration and Value Iteration, two canonical dynamic programming solutions. We then introduce equivalent formulations of the aforementioned solutions for continuous $\mathcal{S}$ MDPs. Global exact solutions to discrete $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$ POMDPs are presented next, with a focus on Monahan’s formulation of exact Value Iteration and an associated Point-based solution. Mathematical formulations of solution techniques for continuous $\mathcal{S}$, continuous $\mathcal{A}$ and $\mathcal{O}$ cases are presented next. We conclude by introducing *local* solution techniques for discrete state $\mathcal{S}$ MDPs and POMDPs that develop a policy for a subset of the complete state space $\mathcal{S}$.

3.1 Markov Decision Processes

Solving an MDP means finding the optimal policy π^* which, in the case of finite $\mathcal{S}$ and $\mathcal{A}$ MDPs, exists in a policy space with $|\mathcal{S}|^{|\mathcal{A}|}$ possible policies. There are two broad approaches for solving MDPs: (i) model-based methods; and (ii) model-free methods. Model based methods assume that the components of the MDP are fully specified, whereas model-free methods make no such assumption and instead estimate a model of the environment while it is learning. In the following sections

Algorithm 1: PolicyEvaluation(MDP, $|\mathcal{D}|$, γ , ϵ , π) $\longrightarrow \langle V_d \rangle$

```

1 begin
2    $d := 0, V_d := \{0\}$ 
3   while  $d < |\mathcal{D}|$  do
4      $d := d + 1$ 
5     foreach  $s \in \mathcal{S}$  do
6        $V_d^\pi(s) := \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') [\mathcal{R}(s, \pi(s), s') + \gamma \cdot V_{d-1}^\pi(s')]$ 
7       // Compute the residual
8        $r_d(s) := ||V_d^\pi(s) - V_{d-1}^\pi(s)||$ 
9       // Terminate if early convergence
10      if  $\max r_d(s) < \epsilon$  then
11        break
12  return  $\langle V_d \rangle$ 

```

we limit our discussion to model-based solution techniques.

3.1.1 Discrete $\mathcal{S}$

In this section we introduce two well known exact solution methods, policy iteration and value iteration, for MDPs with *discrete* state $\mathcal{S}$ and action $\mathcal{A}$ spaces.

Policy Iteration

Policy iteration (PI) [Howard, 1960] finds an intelligent order for exploring the space of $|\mathcal{S}|^{|\mathcal{A}|}$ policies such that the current policy is guaranteed to be better than all of the previously considered policies. As a result PI typically avoids enumerating all $|\mathcal{S}|^{|\mathcal{A}|}$ policies. Policy iteration achieves this by alternating between policy evaluation and policy improvement until there is convergence to an optimal policy π^* , which we introduce in the proceeding sections.

Policy Evaluation Policy evaluation finds the value function V^π of an arbitrary fixed policy π . V^π can be specified as a system of $|\mathcal{S}|$ linear equations with $|\mathcal{S}|$ unknowns. Whilst this solution is straightforward it can be tedious to compute. An alternate method is to use an iterative procedure to calculate a sequence of approximate value functions V_d^π that converge to V^π as $d \rightarrow \infty$. V_0 is chosen arbitrarily and each V_d^π is computed from the previous V_{d-1}^π using Equation (2.4) as an update rule:

$$V_d^\pi(s) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') [\mathcal{R}(s, \pi(s), s') + \gamma \cdot V_{d-1}^\pi(s')]. \quad (3.1)$$

A backup for a single state s requires accessing the values of all the neighbours, which is an $\mathcal{O}(|\mathcal{S}|)$ operation. Thus, a full backup over all $s \in \mathcal{S}$ runs in $\mathcal{O}(|\mathcal{S}|^2)$ time. Convergence to V^π may take infinitely many iterations.

Algorithm 2: PolicyIteration(MDP, $|\mathcal{D}|$, γ , ϵ) $\longrightarrow \langle \pi_d \rangle$

```

1 begin
2    $d := 0, \pi_d := \{0\}$ 
3   while  $d < |\mathcal{D}|$  do
4      $d := d + 1$ 
5     // Terminate if no improvement
6     if  $\pi_d = \pi_{d-1}$  then
7       break
8     // Policy Evaluation
9      $V_d^{\pi_d} := \text{PolicyEvaluation}(\text{MDP}, |\mathcal{D}|, \gamma, \epsilon, \pi_d)$ 
10    // Policy Improvement
11    foreach  $s \in \mathcal{S}$  do
12       $\pi_d(s) := \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') [\mathcal{R}(s, a, s') + \gamma \cdot V_{d-1}(s')]$ 
13  return  $\langle \pi_d \rangle$ 

```

Policy Improvement Policy improvement attempts to improve the policy V^π , the outcome of the Policy evaluation stage. For each state $s \in \mathcal{S}$, policy improvement determines whether it is better to choose an action $a \neq \pi(s)$. This premise can be evaluated via:

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') [\mathcal{R}(s, a, s') + \gamma \cdot V^\pi(s')]. \quad (3.2)$$

A *greedy* policy is calculated from Equation (3.2) by selecting the best action in each state:

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^\pi(s, a). \quad (3.3)$$

Policy improvement gives a strictly better policy except when the original policy is already optimal.

Value Iteration

Value Iteration (VI) computes an MDP solution by directly solving the Bellman optimality equation [Bellman, 1957]. The key idea is to successively approximate V^* with a V_d function, such that V_d converges to V^* as $d \rightarrow \infty$. Algorithm (3) outlines the procedure for this algorithm.

VI starts with an initial value of $V_d = 0$ and performs lines 4-12 for each epoch d . At each epoch V_d is calculated from V_{d-1} as follows:

$$V_d(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') [\mathcal{R}(s, a, s') + \gamma \cdot V_{d-1}(s')]. \quad (3.4)$$

Equation (3.4) corresponds to lines 7-8 in Algorithm (3).

VI is a method to approach the fixed-point of the Bellman equations using suc-

Algorithm 3: ValueIteration(MDP, $|\mathcal{D}|$, γ , ϵ) $\longrightarrow \langle V_d, \pi_d \rangle$

```

1 begin
2    $d := 0, V_d := \{0\}$ 
3   while  $d < |\mathcal{D}|$  do
4      $d := d + 1$ 
5     foreach  $s \in \mathcal{S}$  do
6       foreach  $a \in \mathcal{A}$  do
7         // Bellman backup
7          $Q_d^a(s, a) := \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') [\mathcal{R}(s, a, s') + \gamma \cdot V_{d-1}(s')]$ 
8          $V_d(s) := \max_{a \in \mathcal{A}} Q_d^a(s, a)$ 
9          $\pi_d(s) := \operatorname{argmax}_{a \in \mathcal{A}} Q_d^a(s, a)$ 
10      // Terminate if early convergence
11      if  $\|V_d - V_{d-1}\| < \epsilon$  then
12        break
13  return  $\langle V_d, \pi_d \rangle$ 

```

cessive approximation. A single Bellman backup requires iterating over all actions and all successor states, which in the worst case runs in $O(|\mathcal{S}||\mathcal{A}|)$. A complete iteration requires $O(|\mathcal{S}^2||\mathcal{A}|)$ [Papadimitriou and Tsitsiklis, 1987]. Beyond this number of iterations V_d becomes closer to V^* but the corresponding policy π ceases to change.

In practice, several stopping criteria can be used to terminate the algorithm. The criterion used in Algorithm (3), $\|V_d - V_{d-1}\| < \epsilon$, terminates when the distance between V_d and V_{d-1} is smaller than an *a priori* threshold ϵ . The convergence complexity of VI can be further improved by utilising asynchronous dynamic programming [Bertsekas and Tsitsiklis, 1989], where the order in which states are updated are selected at random, or by pruning sub-optimal actions.

Linear Programming

The set of Bellman equations for a finite MDP can also be solved using the following linear programming formulation [d'Epenoux, 1963]:

$$\begin{aligned}
& \text{Minimise} && \sum_{s \in \mathcal{S}} \alpha(s) \cdot V^*(s) \\
& \text{Subject to} && V^*(s) \geq \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') [\mathcal{R}(s, a, s') + \gamma \cdot V(s')] \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}
\end{aligned}$$

The $\alpha(s)$ represents a set of weights such that $\alpha(s) > 0 \forall s \in \mathcal{S}$. The constraints define a lower bound on the value function; the linear program finds the smallest value that is at least as large as the total expected discounted reward. The linear programming formulation is mathematically equivalent to policy iteration [Puterman, 1994].

3.1.2 Continuous Extensions

In this section we reformulate the exact solution methods presented in Section 3.1.1 for MDPs with continuous state $\mathcal{S}$ spaces. We note that even though the formalisation presented in this section is limited, it is useful in motivating the general mixed discrete and continuous factored state case introduced in Chapter 4.

Continuous $\mathcal{S}$

Here we present the mathematical formulation of policy iteration and value iteration for MDPs with *continuous* state $\mathcal{S}$ spaces. We note that integration with respect to the state space $\mathcal{S}$ are specified in their most general form and may not be tractable or known in closed-form.

Policy Iteration

Policy Iteration for continuous $\mathcal{S}$ MDPs can be derived from Equations (3.1) to (3.3).

Policy Evaluation Policy evaluation in continuous $\mathcal{S}$ MDPs can be written as follows:

$$V_d^\pi(s) = \int \mathcal{T}(s, \pi(s), s') [\mathcal{R}(s, \pi(s), s') + \gamma \cdot V_{d-1}^\pi(s')] dS'.$$

Policy Improvement Policy improvement in continuous $\mathcal{S}$ MDPs can be written as follows:

$$Q^\pi(s, a) = \int \mathcal{T}(s, a, s') [\mathcal{R}(s, a, s') + \gamma \cdot V(s')] dS'. \quad (3.5)$$

A greedy policy is calculated from Equation (3.5) via:

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^\pi(s, a). \quad (3.6)$$

Value Iteration

Value Iteration for continuous $\mathcal{S}$ MDPs can be simply stated as follows:

$$V_d(s) = \max_{a \in \mathcal{A}} \int \mathcal{T}(s, a, s') [\mathcal{R}(s, a, s') + \gamma \cdot V_{d-1}(s')] dS'. \quad (3.7)$$

Equation (3.7) is derived from Equation (3.4) by replacing s, s and the Σ operator with their continuous state counterparts.

3.2 Partially Observable Markov Decision Processes

POMDPs generalise MDPs to situations where the state $\mathcal{S}$ is not fully observable. This realistic extension of MDPs has the adverse affect of increasing the complexity of exact POMDP solutions to be almost virtually intractable. Despite these concerns we proceed to introduce algorithms for discrete and continuous $\mathcal{S}$ POMDPs.

3.2.1 Discrete $\mathcal{S}$

In this section we introduce two well known value iteration based solution methods for POMDPs with *discrete* state $\mathcal{S}$, action $\mathcal{A}$ and observation $\mathcal{O}$ spaces.

All POMDP value iteration methods search for Γ , the set of α -vectors or α -functions, that represent the value function at a particular epoch $d \in \mathcal{D}$. Value iteration methods differ in the way they approach building Γ_d from Γ_{d-1} by: (i) generating only useful plans; or (ii) pruning dominated α -vectors or α -functions. We focus on methods that take the latter approach and begin by introducing Monahan's Exact Value Iteration algorithm as the canonical approach.

Exact Value Iteration

In Monahan's formulation of exact VI [Monahan, 1982], the value function at epoch V_d is calculated in terms of the following vector operations:

$$\alpha_{i,d}^{a,o} = \mathcal{R}(s,a) \cdot \frac{1}{|\mathcal{O}|} + \gamma \cdot \sum_{s' \in \mathcal{S}} \mathcal{Z}(o,a,s') \cdot \mathcal{T}(s,a,s') \cdot \alpha_{i,d-1}(s') \quad \forall \alpha_{i,d-1} \in V_{d-1} \quad (3.8)$$

$$\begin{aligned} \Gamma_d^{a,o} &= \bigcup_i \left\{ \alpha_{i,d}^{a,o} \right\} \\ \Gamma_d^a &= \boxplus_{o \in \mathcal{O}} \Gamma_d^{a,o} \\ V_d &= \bigcup_{a \in \mathcal{A}} \Gamma_d^a \end{aligned} \quad (3.9)$$

Equation (3.8) shows that, for a specific action $a \in \mathcal{A}$ and observation $o \in \mathcal{O}$, α -vectors are built from the previous set of α -vectors in Γ_{d-1} . This approach is independent of a particular belief in $\mathcal{P}(\mathcal{S})$ and as such it enumerates all possible α -vectors using a cross-summation operator in Equation (3.9). Equations (3.8) and (3.9) generate $|\mathcal{A}||\mathcal{O}||V_{d-1}|$ and $|\mathcal{A}||V_{d-1}|^{|\mathcal{O}|}$ α -functions respectively, at every epoch d . Hence, in the worst case $|V_d| = |\mathcal{A}||V_{d-1}|^{|\mathcal{O}|}$. Algorithm (4) outlines the procedure for Monahan's exact VI.

Many of the α -vectors that comprise V_d may not affect the optimal policy and can be pruned without affecting solution quality. One such method is to use a Linear Program to detect and prune α -vectors that are dominated [White III and Kim, 1980] i.e. α_i is dominated if $\alpha_i(s) \leq \alpha_j(s) \forall s \in \mathcal{S}$. In theory it is possible to reduce the number of α -functions in V_d without affecting solution quality, however it is intractable for any reasonably sized domain [Cassandra et al., 1997, Papadimitriou and Tsitsiklis, 1987].

The aforementioned limitations of Exact VI motivates the use of algorithms that search for only reachable parts of the belief space $\mathcal{P}(\mathcal{S})$ [Pineau et al., 2003, Spaan and Vlassis, 2005].

Algorithm 4: MonahanValueIteration(POMDP, $|\mathcal{D}|, \gamma, \epsilon) \rightarrow V_d$

```

1 begin
2    $d := 0, V_d := \{0\}$ 
3   while  $d < |\mathcal{D}|$  do
4      $d := d + 1, V_d := \emptyset$ 
5     foreach  $a \in \mathcal{A}$  do
6        $\Gamma_d^a := \emptyset$ 
7       foreach  $o \in \mathcal{O}$  do
8         foreach  $\alpha_{i,d-1} \in V_{d-1}$  do
9            $\alpha_{i,d}^{a,o} := \text{see Equation (3.8)}$ 
10           $\Gamma_d^{a,o} := \Gamma_d^{a,o} \cup \{\alpha_{i,d}^{a,o}\}$ 
11           $\Gamma_d^a = \boxplus_{o \in \mathcal{O}} \Gamma_d^{a,o}$ 
12         $V_d := V_d \cup \{\Gamma_d^a\}$ 
13        // Terminate if early convergence;
14        if  $V^d = V^{d-1}$  then
15          break
16    return  $V^d$ 

```

Point-based Solutions

Point based solution methods rapidly approximate POMDP solutions by: (i) bounding the size of the value function V_d by collecting a set of reachable belief points and maintaining a value function only over these points [Pineau et al., 2003]; and (ii) using a compact point-based Bellman backup operation for each $b \in \mathcal{B}$. The general assumption underlying these methods is the resulting policy will generalise well for beliefs outside the set $\mathcal{B}$.

Point Based Value Iteration (PBVI) [Pineau et al., 2003] for POMDPs with discrete and finite $\mathcal{S}, \mathcal{A}$ and $\mathcal{O}$ can be written as follows:

$$\alpha_{i,d}^{a,o} = \mathcal{R}(s, a) \cdot \frac{1}{|\mathcal{O}|} + \gamma \cdot \sum_{s' \in \mathcal{S}} \Omega(o, a, s') \cdot \mathcal{T}(s, a, s') \cdot \alpha_{i,d-1}(s'), \quad \forall \alpha_{i,d-1} \in V_{d-1} \quad (3.10)$$

$$\begin{aligned} \Gamma_d^{a,o} &= \bigcup_i \{\alpha_{i,d}^{a,o}\} \\ \Gamma_{b,d}^a &= \sum_{o \in \mathcal{O}} \operatorname{argmax}_{\alpha \in \Gamma_d^{a,o}} (\alpha \cdot b) \end{aligned} \quad (3.11)$$

$$\alpha_{b,d} = \operatorname{argmax}_{\Gamma_{b,d}^a, a \in \mathcal{A}} (\Gamma_{b,d}^a \cdot b), \quad \forall b \in \mathcal{B} \quad (3.12)$$

$$V_d = \bigcup_{b \in \mathcal{B}} \{\alpha_{b,d}\} \quad (3.13)$$

In the formulation above, the initial value function V_0 is set as it was for Exact VI. PBVI implicitly prunes dominated α -vectors twice via the argmax operation in Equations (3.11) and (3.12), respectively. Thus, the final $|V_d|$ is limited to $|\mathcal{B}|$, which is in stark contrast to exact VI where $|V_d| = |\mathcal{A}| |V_{d-1}|^{|\mathcal{O}|}$. Algorithm (5) outlines the

Algorithm 5: PBVI(POMDP, $|\mathcal{D}|, \gamma, \epsilon) \rightarrow V_d$

```

1 begin
2    $d := 0, V_d := \{0\}$ 
3   while  $d < |\mathcal{D}|$  do
4      $d := d + 1, V^d := \emptyset$ 
5     foreach  $b \in \mathcal{B}$  do
6       foreach  $a \in \mathcal{A}$  do
7          $\Gamma_{b,d}^a := \emptyset, \alpha_{b,d}^a := 0$ 
8         foreach  $o \in \mathcal{O}$  do
9            $\Gamma_d^{a,o} := \emptyset$ 
10          foreach  $\alpha_{i,d-1} \in V_{d-1}$  do
11             $\alpha_{i,d}^{a,o} := \text{see Equation (3.10)}$ 
12             $\Gamma_d^{a,o} := \Gamma_d^{a,o} \cup \{\alpha_{i,d}^{a,o}\}$ 
13             $\alpha_b^{a,o} := \text{argmax}_{\alpha \in \Gamma_d^{a,o}} (\alpha \cdot b)$ 
14             $\alpha_{b,d}^a := \alpha_{b,d}^a + \alpha_b^{a,o}$ 
15             $\Gamma_{b,d}^a := \Gamma_{b,d}^a \cup \{\alpha_{b,d}^a\}$ 
16             $\alpha_{b,d} := \text{argmax}_{a \in \mathcal{A}} (\Gamma_{b,d}^a \cdot b)$ 
17             $V_d := V_d \cup \{\alpha_{b,d}\}$ 
18          // Terminate if early convergence;
19          if  $V_d = V_{d-1}$  then
20            break
21  return  $V_d$ 

```

procedure for PBVI.

In PBVI the backup operation becomes more costly as the number of belief points, b , increases. Perseus [Spaan and Vlassis, 2005] performs the backup operation on a randomly selected subset of belief points $b \in \mathcal{B}$, while ensuring that the value of these points increase with each epoch i.e. $V_d(b) \leq V_{d+1}(b)$. The key idea is to backup a randomly selected subset of $\tilde{\mathcal{B}}$:

$$V_d = \bigcup_{b \in \tilde{\mathcal{B}}} \{\alpha_{b,d}\}.$$

In each backup stage the set $\tilde{\mathcal{B}}$ is constructed by sampling beliefs from $\mathcal{B}$ until the resulting V_d upper bounds V_{d-1} over $\mathcal{B}$:

$$V_{d-1}(b') \leq V_d(b'), \forall b' \in \mathcal{B}.$$

The Perseus operator allows for the use of much larger $\mathcal{B}$ which in turn has a positive effect on the approximation accuracy [Pineau et al., 2003].

Policy Iteration

Policy iteration (PI) [Howard, 1960] finds an intelligent order for exploring the space of policies so that the current policy is guaranteed to be better than all the previous policies considered. In POMDPs policies are defined over the space of beliefs $\mathcal{P}(\mathcal{S})$ which is continuous. Hence, the challenge of finding an optimal policy becomes one of finding the global maximum in a continuous space.

Sondik's PI algorithm [Sondik, 1971, Sondik, 1978] represents a policy as a mapping from a finite number of polyhedral belief space regions to actions. This mapping is subsequently converted to a Finite State Controller (FSC) in order to calculate the policy's value. While this algorithm is, in theory, able to find the exact optimal policy for POMDPs, it is not widely used in practice.

Hansen [Hansen, 1998] extends Sondik's algorithm by directly representing a policy as an FSC. Hansens PI algorithm is analogous to that presented for MDPs in Section 3.1.1 but instead uses a deterministic FSC to represent an initial policy. The algorithm performs the policy evaluation and policy improvement steps, improving the current FSC into an improved one. The sequence of FSC's converge to the optimal policy π^* .

3.2.2 Continuous Extensions

In the previous sections we considered POMDPs with discrete $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$. In this section we introduce solution methods for continuous extensions of the POMDP framework used thus far, that is, for continuous $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$.

Continuous $\mathcal{S}$

POMDPs with continuous $\mathcal{S}$, discrete and finite action $\mathcal{A}$, and observations $\mathcal{O}$ can be solved using an unrestricted formulation of Point Based Value Iteration:

$$\begin{aligned}\alpha_{i,d}^{a,o} &= \mathcal{R}(s,a) \cdot \frac{1}{|\mathcal{O}|} + \gamma \cdot \int \Omega(o,a,s') \cdot \mathcal{T}(s,a,s') \cdot \alpha_{i,d-1}(s') dS', \quad \forall \alpha_{i,d-1} \in V_{d-1} \\ \Gamma_d^{a,o} &= \bigcup_i \{ \alpha_{i,d}^{a,o} \} \\ \Gamma_{b,d}^a &= \sum_{o \in \mathcal{O}} \operatorname{argmax}_{\alpha \in \Gamma_d^{a,o}} (\alpha \cdot b) \\ \alpha_{b,d} &= \operatorname{argmax}_{\Gamma_{b,d}^a, a \in \mathcal{A}} (\Gamma_{b,d}^a \cdot b), \quad \forall b \in \mathcal{B} \\ V_d &= \bigcup_{b \in \mathcal{B}} \{ \alpha_{b,d} \}\end{aligned}$$

The formulation above is derived from that presented in Section 3.2.1. Despite the ease of reformulation, we place particular emphasis on the fact that the calculation of the α -function, which requires integration with respect to the state space $\mathcal{S}$, may not be tractable or known in closed-form if its constituent functions are left in their most general forms. Zamani and Sanner [Zamani et al., 2012b] proved that

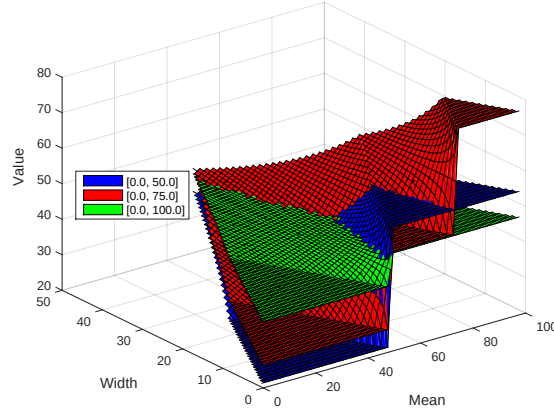


Figure 3.1: An α -function defined with respect to a Uniform belief. The upper surface of the plot indicates the optimal α -function for the value belief defined by $v \sim \mathcal{U}[m - w, m + w]$, for a given mean m and width w .

the α -function remains piecewise linear if the observation function $\mathcal{O}$ is restricted to piecewise constant functions and the transition and reward functions, $\mathcal{T}$ and $\mathcal{R}$, are restricted to piecewise linear functions, respectively. An alternate approach, used by Porta et al. [Porta et al., 2005], restricts all functions defined on the state space to be Gaussian or a finite linear combination of Gaussian functions. Hence, the resulting α -function is also a Gaussian function.

The visualisation of α -functions is considerably more complicated than that of α -vectors. α -functions, when they can be calculated in closed-form, must be visualised as a parameterised function of an assumed univariate distribution. In Figure (3.1) we show an α -function defined with respect to a uniform belief.

Continuous $\mathcal{A}$ and $\mathcal{O}$

POMDPs with continuous $\mathcal{A}$ or $\mathcal{O}$ spaces are more difficult to solve due to the fact that it is no longer possible to define a conditional plan or build policy trees.

For POMDPs with continuous $\mathcal{A}$, maximising over all actions is clearly infeasible. Hence, one must resort to sampling-based techniques. Spaan and Vlassis [Spaan and Vlassis, 2005] replace full maximisation over actions $\mathcal{A}$ with a *sampled* max operator that performs the maximisation over a random subset of $\mathcal{A}$ [Szepesvári and Littman, 1996].

For POMDPs with continuous $\mathcal{O}$, conditional plans are decision trees with infinitely many branches and observation strategies are continuous functions. One solution technique is to dynamically partition the observation space based on a current policy [Hoey and Poupart, 2005]. Policies with a small number of conditional plans can have their observations aggregated over a small number of regions, with each region corresponding to the relevant observation. The region-based observation function is generally constructed by using general sampling techniques.

Zamani and Sanner [Zamani et al., 2012b] extend [Hoey and Poupart, 2005] to

POMDPs with continuous $\mathcal{S}$ and $\mathcal{O}$ spaces with potentially correlated, multivariate continuous observation spaces. The PBVI based algorithm introduced in [Zamani et al., 2012b] derives a set of continuous observation partitions that permit exact point-based backups at a particular belief point. The algorithm restricts the transition function $\mathcal{T}$ to be piecewise linear with discrete noise, the reward function $\mathcal{R}$ to be piecewise constant and the observation probabilities and beliefs to be uniformly distributed.

3.3 Local Solution Techniques

In this section we introduce *local* solution techniques to the frameworks for sequential decision making that were presented in Chapter 2. Local solution techniques aim to develop a policy $\pi : \mathcal{S}' \subseteq \mathcal{S} \rightarrow \mathcal{A}$, where $\mathcal{S}'$ is a subset of the complete state space $\mathcal{S}$. By defining a policy over $\mathcal{S}' \subseteq \mathcal{S}$ cost of calculating the policy for a given domain becomes exponential in the decision epoch $\mathcal{D}$ as opposed to polynomial in $\mathcal{S}$ and $\mathcal{A}$, as is the case when the complete state space $\mathcal{S}$ is used [Kearns et al., 2002]. Monte Carlo Tree Search (MCTS) [Coulom, 2007] and its generalisation Trial-based Heuristic Tree Search (THTS) [Keller and Helmert, 2013] provide a unifying framework for the specific local MDP and POMDP solution techniques that we present herein.

Monte Carlo Tree Search is a family of algorithms used to find optimal actions in a domain by taking random samples in the decision space and building a search tree according to the results. MCTS rests on two fundamental concepts: (i) that the true value of an action may be approximated using random simulation; and (ii) that these values may be used efficiently to adjust the policy towards a best-first strategy.

The basic MCTS algorithm iteratively builds a partial search tree from the root node v_0 , corresponding to state $s_0 \in \mathcal{S}$, until some predefined stopping criterion, at which point the action $a \in \mathcal{A}$ that leads to the *best* child node of the root v_0 is returned. The search tree is built iteratively using two separate policies. The *tree policy* selects or creates a leaf node, with an associated state $s_l \in \mathcal{S}$, from the nodes already contained within the search tree. The *default policy* is used to run a simulation of the domain from the node selected by the tree policy to produce a value estimate of s_l . The results of the default policy are then back-propagated through the selected nodes to update node statistics that inform future tree policy decisions. In the subsequent sections we present a selection of MCTS algorithms to solve MDPs and POMDPs.

3.3.1 Markov Decision Processes

Local solutions to MDPs aim to calculate a policy π that covers a *subset* $\mathcal{S}'$ of the entire state space $\mathcal{S}$. By restricting the set of states that are considered local solution algorithms decrease the cost of calculating an optimal solution from being polynomial in $\mathcal{S} \times \mathcal{A}$, as in the case of global planners, to being exponential in the decision epoch $\mathcal{D}$. Furthermore, the decision epoch $\mathcal{D}$ is under the control of the user, who can trade off optimality with a computational budget.

In the following sections we introduce local solution algorithms for MDPs with large discrete state space $\mathcal{S}$ and action space $\mathcal{A}$.

Discrete $\mathcal{S}$

Local solutions to discrete $\mathcal{S}$ MDPs can be grouped into three main formalisms: (i) heuristic search [Hansen and Zilberstein, 2001]; (ii) dynamic programming [Barto et al., 1995]; and (iii) MCTS. We note that the framework of Trial-based Heuristic Tree Search (THTS) algorithms [Keller and Helmert, 2013] subsumes the three aforementioned formalisms, differentiating them on the basis of five characteristics: (i) heuristic function; (ii) backup function; (iii) action selection; (iv) outcome selection; and (v) trial length. We forego further exploration of the THTS framework in favour of more popular MCTS algorithms.

The UCT algorithm [Kocsis and Szepesvári, 2006] is the most popular representative of MCTS. UCT uses a modified UCB1 bandit [Auer et al., 2002] for the tree policy within the MCTS scheme. The discrete $\mathcal{A}$ in UCB1 correspond to $\mathcal{A}$ of the MDP and the rewards correspond to the accumulated rewards accrued by the default policy executed from a given node.

At any round n the UCT algorithm starts at the root and continues to the leaves by taking a path according to the maximal *B-value* score between the child nodes. The B-value for any child k of node j is calculated by:

$$B_n(k) = \hat{\mu}_{k,n} + c \cdot \sqrt{\frac{\ln T_k(n)}{T_k(n)}},$$

where $T_k(n)$ represents the number of paths that went through node k up to time n and $\hat{\mu}_{k,n}$ is the empirical average of rewards obtained from leaves originating from node k .

A significant benefit of UCT is that it does not require a heuristic to evaluate leaf nodes, which means that it is readily applicable to many domains. Furthermore, UCT is *consistent*, that is, the probability of selecting the optimal action converges to 1.0 as the number of samples grows to infinity.

3.3.2 Partially Observed Markov Decision Processes

Solutions to POMDPs must overcome the *curse of dimensionality* and the *curse of history* [Pineau et al., 2003]. The former arises from the fact that the state space $\mathcal{S}$ is not fully observable, which means that an agent must reason with beliefs, which are probability distributions over the states. The latter indicates that the number of action-observation histories that must be considered increases exponentially with the decision epoch $\mathcal{D}$. Furthermore, the decision epoch $\mathcal{D}$ is under the control of the user, who can trade off optimality with a computational budget. There are two main approaches to POMDP planning: (i) offline policy computation; and (ii) online search. Offline planning involves first calculating a policy contingent upon *all future* scenarios followed by policy execution. By contrast, online planning interleaves

planning and execution. There are three main categories of online planning algorithms: (i) heuristic search; (ii) branch-and-bound pruning; and (iii) Monte Carlo sampling [Ross et al., 2008]. In the following sections we introduce local solution algorithms for POMDPs with large discrete state space $\mathcal{S}$ and action space $\mathcal{A}$ that utilise Monte Carlo sampling.

Discrete $\mathcal{S}$

In our discussion of local solutions to POMDPs with discrete $\mathcal{S}$ and $\mathcal{A}$ we confine our focus to a canonical algorithm based on the MCTS paradigm. Partially Observable Monte-Carlo Planning (POMCP) [Silver and Veness, 2010] extends the MCTS framework by introducing a modified UCT search algorithm and a particle filter that updates the agents belief state.

Partially Observable UCT (PO-UCT) extends the standard UCT algorithm by considering *histories* instead of states. Each node in PO-UCT represents a unique history and an associated count for the number of times that history has been visited and the value of that history, determined by the mean return of all simulations starting from that history.

The second innovation of POMCP is to approximate the belief state using an un-weighted particle filter. The belief state for a history is approximated by K particles, where each particle corresponds to a sample state. The belief state is taken to be the sum of all K particles. The K particles are updated using a Monte-Carlo procedure based on sample observations, rewards and transitions.

POMCP combines Monte-Carlo belief state updates with PO-UCT, and shares the same simulations for both Monte-Carlo procedures, making POMCP a computationally efficient algorithm.

3.4 Summary

Exact solution techniques to Markov Decision Processes and Partially Observable Markov Decision Processes aim to calculate a policy over the *complete* state space $\mathcal{S}$. In this chapter we presented algorithms to solve MDPs and POMDPs with discrete and continuous $\mathcal{S}$, discrete $\mathcal{A}$, discrete $\mathcal{O}$ and finite $\mathcal{D}$.

Optimal policies for Markov Decision Processes with discrete $\mathcal{S}$ can be calculated using either dynamic programming or linear programming. The former technique encapsulates two fundamental algorithms, policy iteration and value iteration, respectively. Policy iteration finds an intelligent order for exploring the space of $|\mathcal{S}|^{|\mathcal{A}|}$ policies such that the current policy is guaranteed to be better than all of the previously considered policies. As a result PI typically avoids enumerating all $|\mathcal{S}|^{|\mathcal{A}|}$ policies. Value Iteration computes an MDP solution by directly solving the Bellman optimality equation. VI achieves this by successively approximating V^* with V_d , such that V_d converges to V^* as $d \rightarrow \infty$. Finally, we presented a linear programming solution which is mathematically equivalent to policy iteration. We also introduced the mathematical formulation of policy iteration and value iteration for MDPs with

continuous state $\mathcal{S}$ spaces. We leave the explicit details surrounding their solution to subsequent chapters.

Calculating optimal policies for POMDPs with discrete $\mathcal{S}$ is complicated by the fact that the state $\mathcal{S}$ is not fully observable. We presented two dynamic programming techniques to calculate an optimal policy for a POMDP: (i) value iteration; and (ii) policy iteration. Monahan’s algorithm was presented as the canonical example of value iteration for POMDPs which, in the worst case, requires $|\mathcal{A}||V_{d-1}|^{|\mathcal{O}|}$ α -vectors to represent V_d , the value function at epoch d . Point based solutions, such as Point Based Value Iteration, alleviate this problem by bounding the size of the value function V_d by defining it for a set of reachable belief points. Policy Iteration methods for POMDPs rely on a finite state controller representation of a policy, which is then used within the familiar policy evaluation and policy improvement cycle, to converge to the optimal policy. We also introduced the mathematical formulation of Point Based Value Iteration for MDPs with *continuous* state $\mathcal{S}$ spaces. However, we leave the explicit details surrounding their solution to subsequent chapters. Continuous action $\mathcal{A}$ and/or observation $\mathcal{O}$ POMDPs face the further challenge of not being able to define conditional plans or policy trees.

Local solution techniques mitigate computational intractability of exact solution techniques for large MDPs and POMDPs by restricting their solutions to a subset of the state space $\mathcal{S}' \subseteq \mathcal{S}$. Monte Carlo Tree Search was presented as a family of algorithms used to calculate local solutions to both MDPs and POMDPs. MCTS rests on two fundamental concepts: (i) that the true value of an action may be approximated using random simulation; and (ii) that these values may be used efficiently to adjust the policy towards a best-first strategy. The UCT algorithm is the most popular MCTS algorithm for MDPs and relies on the UCB1 bandit for its tree policy. In the POMDP setting, POMCP is the seminal application of the MCTS paradigm to POMDPs. POMCP utilises a partially observable extension of UCT, known as PO-UCT, as its tree policy and utilises a particle filter to approximate the belief state. MCTS based algorithms present an exciting avenue for further research into sequential decision making in markets and will be elaborated upon in Section 8.2.

In the proceeding chapters we demonstrate how we can calculate optimal policies for continuous $\mathcal{S}$ formulations of MDPs and POMDPs by using Symbolic Dynamic Programming.

Structured Discrete and Continuous Solutions

In this chapter we introduce structured discrete and continuous solutions to the sequential decision making frameworks introduced in Chapter 2. The *exact* solution techniques presented in Chapter 3 are efficient for only moderately sized MDPs and POMDPs with discrete $\mathcal{S}$. Increasing the size of the state space $\mathcal{S}$ leads to the curse of dimensionality [Bellman, 1957]. One attempt to mitigate the adverse affects on memory and space is to exploit the internal structure of MDPs and POMDPs, which for example, may exist in the transition function $\mathcal{T}$. Internal structure allows one to compactly model and solve MDPs and POMDPs.

We begin our discussion with an overview on representations for structured MDPs and POMDPs and their corresponding solution techniques. We then extend the discussion to frameworks with continuous states $\mathcal{S}$ by introducing Symbolic Dynamic Programming (SDP). Two key components of SDP, case statements and their operations, are discussed. This is followed by an illustration of how SDP can be applied to a financial market domain. We conclude by introducing three context-specific structures that can be used to solve structured MDPs and POMDPs. We pay particular attention to Extended Algebraic Decision Diagrams (XADDs), a data structure that allows the efficient representation and manipulation of case statements for use within SDP solutions.

4.1 Structured MDPs and POMDPs

The MDP and POMDP solution methods presented thus far represent their domains via a set of states $\mathcal{S}$, with actions $a \in \mathcal{A}$ and, in the case of POMDPs, observations $o \in \mathcal{O}$, inducing a stochastic transition from one state s to another s' . This representation is severely limited in its application to real-world domains, many of which are characterised by large discrete or continuous state spaces $\mathcal{S}$. In this section we describe how the *internal structure* of these frameworks can be exploited to yield a more compact representation [Koller and Parr, 1999].

4.1.1 Structured Representations

The transition, reward and observation functions of MDPs and POMDPs can be represented using Dynamic Bayesian Networks (DBNs) [Dean and Kanazawa, 1989]. A DBN exploits the fact that transitions of a variable, be it state or observation, do not usually depend on all other variables. States and observations are described implicitly as an assignment of values to some set of state variables. In the proceeding sections we present structured representations of the relevant elements of the MDP and POMDP frameworks. All other elements of the MDP and POMDP frameworks are assumed to be in their enumerated forms.

State space $\mathcal{S}$

The state space $\mathcal{S}$ is specified via a set of random variables $\mathcal{X} = \{\mathcal{X}_1, \dots, \mathcal{X}_n\}$, where each $\mathcal{X}_i$ takes on values in a finite, countably infinite, or uncountable domain $\text{Dom}(\mathcal{X}_i)$. A state x defines a value $x_i \in \text{Dom}(\mathcal{X}_i)$ for each variable $\mathcal{X}_i$. Therefore, the state space $\mathcal{S}$ can be defined by $\text{Dom}(\mathcal{X})$.

The factored state space representation allows one to exploit any structure that may exist in the MDP specification. In particular, a factored state space may allow for the decomposition of the transition $\mathcal{T}$ and reward $\mathcal{R}$ functions via a property, known as context-specific independence, which specifies that the transition and reward functions may not depend on all states or actions. This in turn allows the MDP to be represented more compactly and reduces its computational complexity.

Observation space $\mathcal{O}$

The observation space $\mathcal{O}$ is specified via a set of random variables $\mathcal{Y} = \{\mathcal{Y}_1, \dots, \mathcal{Y}_m\}$, where each $\mathcal{Y}_i$ takes on values in a finite domain $\text{Dom}(\mathcal{Y}_i)$. A state y defines a value $y_i \in \text{Dom}(\mathcal{Y}_i)$ for each variable $\mathcal{Y}_i$. Therefore, the observation space $\mathcal{O}$ can be defined by $\text{Dom}(\mathcal{Y})$.

Transition function $\mathcal{T}$

The transition function is described by a DBN, which is better characterised as a two-layer Directed Acyclic Graph (DAG) whose nodes consist of the variables at the current epoch and the next.

The DBN captures the effect of an action $a \in \mathcal{A}$ on the next state $\mathcal{X}'_i$ given its parents $\text{Pa}(\mathcal{X}_i)$. Each state $\mathcal{X}'_i$ is associated with a Conditional Probability Function (CPF) $\mathbb{P}(\mathcal{X}'_i | \{\text{Pa}(\mathcal{X}_i)\}, a)$. We assume that there are no *synchronic arcs* that specify dependencies between next state variables. As a result, the joint conditional probability distribution can be defined as a product of the CPFs for next state variables,

$$\mathbb{P}(\mathcal{X}' | \mathcal{X}, a) = \prod_i^n \mathbb{P}(\mathcal{X}'_i | \{\text{Pa}(\mathcal{X}_i)\}, a). \quad (4.1)$$

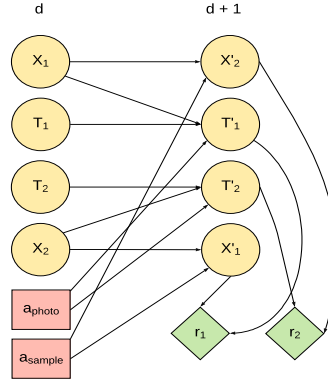


Figure 4.1: DBN representation for the transition and reward functions of a simplified factored Mars Rover. For any action a , a state variable $\mathcal{X}_i$ is conditionally independent of its non-descendents given its parents $\text{Pa}(\mathcal{X}_i)$.

To clarify the factored representation, we briefly consider the canonical Mars Rover domain [Bresina et al., 2002]. In the Mars Rover domain, a rover must approach one or more points of interest and perform actions such as taking a photograph or collecting a sample. Each action executed by the Rover takes time and energy. The Rover must find an optimal path to perform its tasks and minimise its time and energy costs. In Figure (4.1) we depict the transition and reward function of a simplified Mars Rover problem with two coordinates, $\mathcal{X}_1$ and $\mathcal{X}_2$, and two boolean variables for its actions, as a DBN. The DBN captures the effect of an action on a state $\mathcal{X}'_i$ given its parents $\text{Pa}(\mathcal{X}_i)$ where, in the worst case, the next state depends on all variables $\{\mathcal{X}_1, \dots, \mathcal{X}_n\}$ as its parents.

In structured MDPs and POMDPs where the state space $\mathcal{S}$ consists of a mixture of both boolean and continuous variables, the transition function $\mathcal{T}$ permits discrete transition noise. For instance, if $\mathcal{S}$ specifies the state space $(\vec{d}, \vec{x}) = (d_1, \dots, d_m, x_1, \dots, x_n)$, where each $d_i \in \{0, 1\}$ ($1 \leq i \leq m$) is discrete and each $x_j \in \mathbb{R}$ ($1 \leq j \leq n$) is continuous and $\mathcal{A}_s^d$ specifies a finite set of state and epoch dependent actions, the joint transition model can be written as:

$$\mathcal{T} : \mathbb{P}(\vec{d}', \vec{x}' | \vec{d}, \vec{x}, a) = \prod_{i=1}^m \mathbb{P}(d'_i | \vec{d}, \vec{x}, a) \prod_{j=1}^n \mathbb{P}(x'_j | \vec{d}, \vec{d}', \vec{x}, a)$$

where $a \in \mathcal{A}_s^d$. The transition model permits discrete noise in the sense that $\mathbb{P}(x'_j | \vec{d}, \vec{d}', \vec{x}, a)$ may condition on $\vec{d}'$, which are stochastically sampled according to their conditional probability functions [Zamani et al., 2012a, Sanner et al., 2011]. This discrete noise representation allows for the modelling of continuous variable transitions as a mixture of δ functions, which has been used frequently in previous exact continuous state MDP solutions [Feng et al., 2004, Meuleau et al., 2009].

Observation function $\mathcal{Z}$

The observation function of a POMDP is described by a DBN. The DBN captures the effect of an action $a \in \mathcal{A}$ on the next observation $\mathcal{Y}'_i$ given the state variable changes $\mathcal{X}'$. The joint conditional probability distribution can be written as,

$$\mathbb{P}(\mathcal{Y}|\mathcal{X}') = \prod_i^m \mathbb{P}(\mathcal{Y}'_i|\mathcal{X}'). \quad (4.2)$$

Reward function $\mathcal{R}$

The reward function is assumed to be factored additively into a set of localised reward functions, each of which only depend on a small set of variables [Guestrin et al., 2003]. More precisely, a reward function $\mathcal{R}_i$ associated with the state x is restricted to a group of variables $\mathcal{W}_i \subset \{\mathcal{X}_1, \dots, \mathcal{X}_n\}$ such that $\mathcal{R}_i : \text{Dom}(\mathcal{W}_i) \rightarrow \mathbb{R}$. The complete reward function associated with the state x is then defined to be

$$\mathcal{R}(\mathcal{X}', a, \mathcal{X}) = \sum_i^r \mathcal{R}_i(x). \quad (4.3)$$

4.1.2 Structured Value Iteration

Structured Value Iteration extends the Value Iteration algorithm from Section 3.1 with structured representations of the transition and reward functions. The iterative Bellman equation presented in Equation (2.4) can be redefined according to the structured transition $\mathcal{T}$ and reward $\mathcal{R}$ functions as:

$$V_d(\mathcal{X}) = \max_{a \in \mathcal{A}} \sum_{\mathcal{X}'} \prod_i^n \mathbb{P}(\mathcal{X}'_i | \{\text{Pa}(\mathcal{X}_i)\}, a) [\mathcal{R}(\mathcal{X}', a, \mathcal{X}) + \gamma \cdot V_{d-1}(\mathcal{X}')].$$

4.2 Symbolic Dynamic programming

Symbolic Dynamic programming (SDP) [Boutilier et al., 2001] is the process of performing dynamic programming via symbolic manipulations. SDP provides symbolic abstractions for all functions and allows for the logical partitioning of a state space $\mathcal{S}$, continuous or otherwise.

4.2.1 Case Statements

Symbolic Dynamic Programming assumes that all functions can be represented in case statement form [Sanner et al., 2011] as follows:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases}$$

ϕ_i	f_i	Name	Abbreviation
linear	piecewise constant	linear piecewise constant	LPWC
linear	piecewise linear	linear piecewise linear	LPWL
linear	piecewise quadratic	linear piecewise quadratic	LPWQ
nonlinear	piecewise nonlinear	nonlinear piecewise nonlinear	NPWN

Table 4.1: Symbolic Functions used in this Thesis.

The ϕ_i are logical formulae defined over the hybrid state $(\vec{d}, \vec{x})$ where $d_i \in \{0, 1\}$ ($1 \leq i \leq m$) is discrete and $\vec{x} \in \mathbb{R}^n$, that can consist of arbitrary logical combinations of boolean variables and linear inequalities ($\geq, >, <, \leq$) over the continuous variables $\vec{x}$. An example of a simple function represented in case form is as follows:

$$f = \begin{cases} \neg d \wedge (x_1 + x_2 > 5) : & x_1^2 \\ \neg d \wedge (x_1 + x_2 \leq 5) : & x_1 + x_2 \end{cases}$$

The set of conditions $\{\phi_1, \dots, \phi_k\}$ are disjoint. However, ϕ_i may not exhaustively cover the state space $\mathcal{S}$. Hence, f may only be a partial function for some variable assignments. In general, the f_i may be polynomials of $(\vec{d}, \vec{x})$ with non-negative exponents. However, in this thesis we restrict the forms of ϕ_i and f_i to be the combinations listed in Table (4.1).

In the next section we present the main case operations required for SDP.

4.2.2 Case Operations

Scalar Multiplication Scalar multiplication is defined as a unary operation on a single case statement f is defined as:

$$c \cdot f = \begin{cases} \phi_1 : & c \cdot f_1 \\ \vdots & \vdots \\ \phi_k : & c \cdot f_k \end{cases}$$

where $c \in \mathbb{R}$ is applied to each f_i ($1 \leq i \leq k$). A case statement f can be negated by setting $c = -1$.

Binary Operations Binary operations such as addition, subtraction and multiplication are executed in two stages. Firstly, the cross-product of the logical partitions of each case statement is taken, producing paired partitions. Secondly, the binary operation is applied to the resulting paired partitions. The “cross-sum” $\oplus$ operation can be performed on two cases in the following manner:

$$\left\{ \begin{array}{l} \phi_1 : f_1 \\ \phi_2 : f_2 \end{array} \right\} \oplus \left\{ \begin{array}{l} \psi_1 : g_1 \\ \psi_2 : g_2 \end{array} \right\} = \left\{ \begin{array}{ll} \phi_1 \wedge \psi_1 : & f_1 + g_1 \\ \phi_1 \wedge \psi_2 : & f_1 + g_2 \\ \phi_2 \wedge \psi_1 : & f_2 + g_1 \\ \phi_2 \wedge \psi_2 : & f_2 + g_2 \end{array} \right.$$

“cross-subtraction” $\ominus$ and “cross-multiplication” $\otimes$ are defined in a similar manner, but with the addition operator replaced by the subtraction and multiplication operators, respectively. Some partitions resulting from case operators may be inconsistent, that is $\phi_i \wedge \phi_j \models \perp$, and are removed.

Case Minimisation and Maximisation Minimisation over cases, known as *casemin*, is defined as:

$$\text{casemin} \left(\left\{ \begin{array}{l} \phi_1 : f_1 \\ \phi_2 : f_2 \end{array} \right\}, \left\{ \begin{array}{l} \psi_1 : g_1 \\ \psi_2 : g_2 \end{array} \right\} \right) = \left\{ \begin{array}{ll} \phi_1 \wedge \psi_1 \wedge f_1 < g_1 : & f_1 \\ \phi_1 \wedge \psi_1 \wedge f_1 \geq g_1 : & g_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 < g_2 : & f_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 \geq g_2 : & g_2 \\ \phi_1 \wedge \psi_1 \wedge f_2 < g_1 : & f_1 \\ \phi_1 \wedge \psi_1 \wedge f_2 \geq g_1 : & g_1 \\ \phi_1 \wedge \psi_2 \wedge f_2 < g_2 : & f_2 \\ \phi_1 \wedge \psi_2 \wedge f_2 \geq g_2 : & g_2 \end{array} \right.$$

casemin preserves the linearity of the constraints and the constant or linear nature of the f_i and g_i .

Maximisation over cases, known as *casemax*, is defined in a similar manner:

$$\text{casemax} \left(\left\{ \begin{array}{l} \phi_1 : f_1 \\ \phi_2 : f_2 \end{array} \right\}, \left\{ \begin{array}{l} \psi_1 : g_1 \\ \psi_2 : g_2 \end{array} \right\} \right) = \left\{ \begin{array}{ll} \phi_1 \wedge \psi_1 \wedge f_1 > g_1 : & f_1 \\ \phi_1 \wedge \psi_1 \wedge f_1 \leq g_1 : & g_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 > g_2 : & f_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 \leq g_2 : & g_2 \\ \phi_1 \wedge \psi_1 \wedge f_2 > g_1 : & f_2 \\ \phi_1 \wedge \psi_1 \wedge f_2 \leq g_1 : & g_1 \\ \phi_1 \wedge \psi_2 \wedge f_2 > g_2 : & f_2 \\ \phi_1 \wedge \psi_2 \wedge f_2 \leq g_2 : & g_2 \end{array} \right.$$

casemax preserves the linearity of the constraints and the constant or linear nature of the f_i and g_i .

If the f_i or g_i are quadratic then the expressions $f_i > g_i$ or $f_i \leq g_i$ will be at most univariate quadratic. Any such constraint can be linearised into a combination of at most two linear inequalities by completing the square, for example, $-x^2 + 20x - 96 > 0 \equiv [x - 10]^2 \leq 4 \equiv [x > 8] \wedge [x \leq 12]$.

Restriction A case statement f can be restricted to cases that satisfy a particular ϕ as follows:

$$f|_{\phi} = \begin{cases} \phi_1 \wedge \phi : & f_1 \\ \vdots & \vdots \\ \phi_k \wedge \phi : & f_k \end{cases}$$

The resulting case statement $f|_{\phi}$ is a partial function, since it is only defined when ϕ holds.

Substitution The substitution operation takes a set of variables θ and their substitutions and applies them to the case statement f . For example, for $\theta = \{x'_1/(x_1 + x_2), x'_2/x_1^2 \exp(x_2)\}$, the left-hand side of the / represents the substitution variable and the right-hand side of the / represents the expression that should be substituted in its place.

θ can be applied to both non-case functions f_i and case partitions ϕ_i as $f_i\theta$ and $\phi_i\theta$, respectively. Substitution of θ into a non-case function $f_i = x'_1 + x'_2$ can be written as $f_i\theta = x_1 + x_2 + x_1^2 \exp(x_2)$. Substitutions into case partitions are performed by applying θ to each inequality operand:

$$f = \begin{cases} x'_1 \leq \exp(x'_2) : & x'_1 + x'_2 \\ x'_1 > \exp(x'_2) : & x'^2_1 + x'^2_2 \end{cases}$$

$$f\theta = \begin{cases} x_1 + x_2 \leq \exp(x_1^2 \exp(x_2)) : & x_1 + x_2 + x_1^2 \exp(x_2) \\ x_1 + x_2 > \exp(x_1^2 \exp(x_2)) : & (x_1 + x_2)^2 - 3x_1^2 \exp(x_2) \end{cases}$$

In general, substitution into case statements can be written as:

$$f\theta = \begin{cases} \phi_1\theta : & f_1\theta \\ \vdots & \vdots \\ \phi_k\theta : & f_k\theta \end{cases}$$

Continuous Maximisation A case statement can be maximised with respect to a continuous variable y as $g(\vec{b}, \vec{x}) = \max_y f(\vec{b}, \vec{x}, y)$. We can rewrite $f(\vec{b}, \vec{x}, y)$ as:

$$\max_y f(\vec{b}, \vec{x}, y) = \text{casemax}_i \max_y \begin{cases} \phi_i(\vec{b}, \vec{x}, y) : & f_i(\vec{b}, \vec{x}, y) \\ \neg \phi_i(\vec{b}, \vec{x}, y) : & -\infty \end{cases}$$

It is possible to compute $\max_y$ for each case partition individually by leveraging two facts: (i) mutually disjoint partitions of f ; and (ii) exploiting the commutativity of $\max_y$ and casemax_i [Zamani et al., 2012a]. The case formulation ensures that all illegal values are mapped to $-\infty$, which reduces to $\text{casemax}(\phi_i(\vec{b}, \vec{x}, y) f_i(\vec{b}, \vec{x}, y), -\infty) = \phi_i(\vec{b}, \vec{x}, y) f_i(\vec{b}, \vec{x}, y)$.

Continuous Integration with δ Functions A case statement f can be integrated with respect to a continuous variable x_j as $\int_{x_j=-\infty}^{\infty} f(x_j) \delta[x_j - h(\vec{z})]$. Continuous integration assumes that $h(\vec{z})$ can be defined as a case statement and $\vec{z}$ does not contain x_j . Integration simply triggers the substitution $\sigma = \{x_j/h(\vec{z})\}$ on f [Sanner et al., 2011]:

$$\int_{x_j=-\infty}^{\infty} f(x_j) \delta \left[x_j - \begin{cases} \phi_1 : & h_1 \\ \vdots & \vdots \\ \phi_k : & h_k \end{cases} \right] dx_j = \begin{cases} \phi_1 : & f \{x_j/h(\vec{z})\} \\ \vdots & \vdots \\ \neg\phi_k : & f \{x_j/h(\vec{z})\} \end{cases}$$

For a generic δ with multiple case partitions, each of the partitions of f are replaced with the partitions of δ . This results in a nested case statement, which can be reduced to a non-nested case statement as in the following example:

$$\begin{cases} \phi_1 : & \begin{cases} \psi_1 : & f_{11} \\ \psi_2 : & f_{12} \end{cases} \\ \phi_2 : & \begin{cases} \psi_1 : & f_{21} \\ \psi_2 : & f_{22} \end{cases} \end{cases} = \begin{cases} \phi_1 \wedge \psi_1 : & f_{11} \\ \phi_1 \wedge \psi_2 : & f_{12} \\ \phi_2 \wedge \psi_1 : & f_{21} \\ \phi_2 \wedge \psi_2 : & f_{22} \end{cases}$$

To solidify the operations presented above, consider the example of integrating $f(x_1, x_2, x_3)$ with respect to x_1 with $\delta[x_1 - (2x_2 + x_3)]$:

$$f(x_1, x_2, x_3) = \begin{cases} x_1 \geq 5 : & x_1 + x_2 \\ x_1 < 5 : & x_1 \end{cases}$$

$$\int f(x_1, x_2, x_3) \delta[x_1 - (2x_2 + x_3)] dx_1 = \begin{cases} 2x_2 + x_3 - 5 \geq 0 : & 3x_2 + x_3 \\ 2x_2 + x_3 - 5 < 0 : & 2x_2 + x_3 \end{cases}$$

Continuous integration with δ -functions is the only form of integration required within this thesis, due to the restrictions on the representation of the transition function. As outlined in Section 4.1.1, we assume that the transition function $\mathcal{T}$ does not contain any *synchronic arcs* between variables in the transition DBN. Hence, we can marginalise variables independently and in any order. A more general integral operator exists for linear piecewise polynomials [Sanner and Abbasnejad, 2012]. However, it is not applicable to SDP for MDPs and POMDPs due to successive integration operations yielding higher order case statements that, after a casemax operation, yield nonlinear piecewise polynomials, which ultimately prevent further integration.

Case statements and their operations are implemented using Extended Algebraic Decision Diagrams (XADDs) [Sanner et al., 2011].

4.2.3 SDP for Financial Markets

In this section we demonstrate how SDP can be used to specify the POMDP financial market trading agent presented in Section 2.2.3. We limited our exposition to the transition function $\mathcal{T}$, observation function $\mathcal{Z}$ and reward function $\mathcal{R}$ and note that the other components of the POMDP framework are as previously defined.

The transition function $\mathcal{T}$ for each state variable in $\mathcal{S}$ can be written as:

$$\begin{aligned}\mathcal{T}(i'|i, m, a) &= \delta \left[i' - \begin{cases} i \geq 1 \wedge a = \text{sell} : & i - 1 \\ \text{otherwise} : & i \end{cases} \right] \\ \mathcal{T}(m'|m, i, a) &= \delta [m' - m]\end{aligned}$$

The Dirac function $\delta[\cdot]$ ensures that the transitions are valid conditional probability functions that integrate to 1.0.

The $\mathcal{Z}$ can be specified as follows:

$$\begin{aligned}\mathcal{Z}(\text{higher}|m, i, a) &= \begin{cases} m' \geq m : & 1.0 \\ \text{otherwise} : & 0.0 \end{cases} \\ \mathcal{Z}(\text{lower}|m, i, a) &= \begin{cases} m' < m : & 1.0 \\ \text{otherwise} : & 0.0 \end{cases}\end{aligned}$$

Here the reward function specifies the return to the trading agent contingent upon its actions. $\mathcal{R}$ can be specified as follows:

$$\mathcal{R}(m, i, a) = \begin{cases} d < |\mathcal{D}| \wedge a = \text{sell} : & m_d \\ d = |\mathcal{D}| \wedge i \geq 1 : & g(\mathcal{U}_{|\mathcal{D}|}) \\ \text{otherwise} : & 0 \end{cases}$$

where $d \in \mathcal{D}$ denotes the decision epoch.

The SDP representation of the POMDP trading agent formulation can be solved using the algorithms presented in the proceeding section.

4.3 Structured Solutions

The factored and Dynamic Bayesian Network framework for structured MDPs and POMDPs presented in the previous section provide a more compact and efficient way to describe large domains. However, these formalisations are of little practical value without the ability to perform computations upon them. In this section we briefly introduce three *context-specific* structures: Decision Trees (DTs) [Boutilier et al., 1995]; Algebraic Decision Diagrams (ADDs) [St-Aubin et al., 2000]; and Extended Algebraic Decision Diagrams (XADDs).

4.3.1 Decision Trees and Algebraic Decision Diagrams

A Decision Tree represents a function by partitioning its input space into a number of simpler regions, each of which are associated with an output value. A DT is composed of two elements:

1. Nodes, which may be either internal or external. Internal nodes define the partition of the input space, whereas external nodes represent the value of the function for the partition defined by its parent internal nodes; and
2. Edges. Edges connect a parent internal node to a child internal or external node.

Decision Trees can often represent conditional probability functions much more compactly than a full conditional probability table. As a result, they can be used to represent a structured transition function $\mathcal{T}$, reward function $\mathcal{R}$, policies and value functions. In the case of structured POMDPs, DTs can also be used to represent the observation function $\mathcal{Z}$ [Boutilier and Poole, 1996].

An Algebraic Decision Diagram represents a function $f : \mathbb{B}^n \rightarrow \mathbb{R}$, where $\mathbb{B}^n$ is an n -dimensional boolean test. An ADD is composed of two elements:

1. Nodes, which may be either internal or external. Internal nodes represent a boolean test on a variable from the input space. Each internal node extrudes two edges corresponding to the value True and False. External nodes represent the value of the function for the partition defined by its parent internal nodes; and
2. Edges. Edges connect a parent internal node to a child internal or external node depending on its associated True or False value.

Unlike a DT, an ADD is capable of unifying identical sub-trees into one DAG, thereby yielding a more compact representation of a function [Hoey et al., 1999]. Furthermore, operations on ADDs exploit their DAG structure and fixed variable ordering.

To illustrate the utility of using DT and ADD representations of functions, consider the Boolean function $f(a, b, c)$ defined in Table (4.2). The tabular representation of the function $f(a, b, c)$ can be represented as a DT and ADD, as shown in Figure (4.2). The DT utilises context-specific independence to yield a simplified representation of $f(a, b, c)$. The ADD exploits context-specific independence and shared substructure to yield an even more compact representation of $f(a, b, c)$.

Tree based data structures such as DTs and ADDs can be used within dynamic programming algorithms to partition the state space by representing each partition using a tree-like structure that branches on state variables and assigns values at the leaves. Two such dynamic programming algorithms are: (i) Structured Value Iteration [Boutilier et al., 2001]; and (ii) Structure Policy Iteration [Boutilier et al., 1995]. Stochastic Planning Using Decision Diagrams (SPUDD) [Hoey et al., 1999] uses ADDs as the data structure within a Structured Value Iteration scheme.

a	b	c	$f(a, b, c)$
0	0	0	0.0
0	0	1	0.0
0	1	0	0.0
0	1	1	1.0
1	0	0	0.0
1	0	1	1.0
1	1	0	0.0
1	1	1	1.0

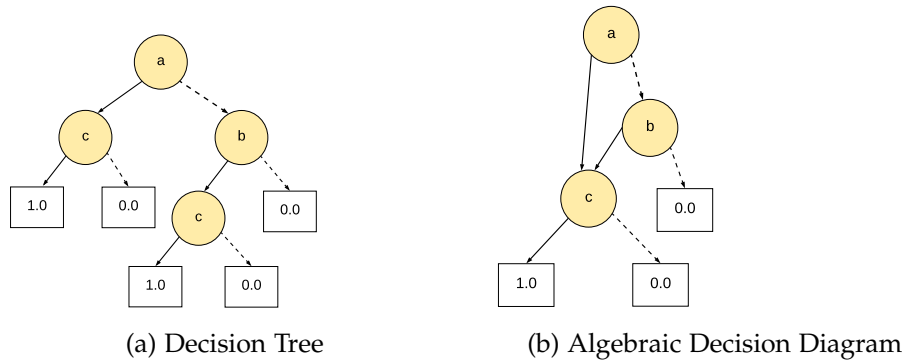
Table 4.2: Tabular representation of a Boolean function $f : \mathbb{B}^3 \rightarrow \mathbb{R}$.

Figure 4.2: Compact tree-based representations of a Boolean function $f : \mathbb{B}^3 \rightarrow \mathbb{R}$ defined in Table (4.2). The Decision Tree exploits context-specific independence, whereas the Algebraic Decision Diagram exploits context-specific independence and shared substructure.

4.3.2 Extended Algebraic Decision Diagrams

Case statements provide a convenient way of representing arbitrary piecewise functions for use within symbolic dynamic programming algorithms. However, their utility in practice is limited when operations on case statements, such as continuous maximisation, result in a case statement that is larger than its operands. The Extended Algebraic Decision Diagram (XADD) data structure was developed as a compact and efficient representation of case statements for use within SDP algorithms.

An XADD is a function represented by a Directed Acyclic Graph (DAG) on $\mathbb{R}$ with the domain $(\vec{d}, \vec{x}) = (b_1, \dots, b_m, x_1, \dots, x_n)$ where $d_i \in \{0, 1\}$ ($1 \leq i \leq m$) is discrete and each $x_j \in [x_j^{\min}, x_j^{\max}]$ ($1 \leq j \leq n$) is continuous with $x_j^{\min}, x_j^{\max} \in \mathbb{R}$ and $x_j^{\min} < x_j^{\max}$. An XADD is composed of two elements:

1. Nodes, which may be either internal or external. Internal decision nodes represent boolean variable tests or continuous inequalities. Each internal node extrudes two edges corresponding to the value True and False. External nodes are arbitrary continuous expressions that represent the value of the function for

the partition defined by its parent internal nodes; and

2. Edges. Edges connect a parent internal node to a child internal or external node depending on its associated `True` or `False` value.

An XADD can be represented as a case statement as follows:

$$f = \begin{cases} \phi_1(\vec{d}, \vec{x}) : f_1(\vec{x}) \\ \vdots \\ \phi_k(\vec{d}, \vec{x}) : f_k(\vec{x}) \end{cases}$$

where f_i are arbitrary expressions over $\vec{x}$ and $\phi_i(\vec{d}, \vec{x})$ are logical formulae over $(\vec{d}, \vec{x})$ defined by arbitrary logical combinations of boolean variables in $\vec{d}$ and linear inequalities ($\geq, >, <, \leq$) over continuous variables $\vec{x}$.

In an XADD each $\phi_i(\vec{d}, \vec{x})$ is represented as an internal decision node leading to the leaf node f_i . The XADD also has a fixed ordering of decision nodes from the root to a leaf where the low (`False`) or high (`True`) outgoing branch of each decision node determines the next node in the XADD.

XADDs nodes, as they have been defined thus far, are able to represent arbitrary expressions. However, throughout this thesis we restrict these expressions to linear functions, which allows us to prune inconsistent linear constraints using an LP-solver, resulting in a more compact XADD.

Figure 4.3 shows how an XADD is used to represent a function f that is piecewise continuous, nonlinear and multi-modal.

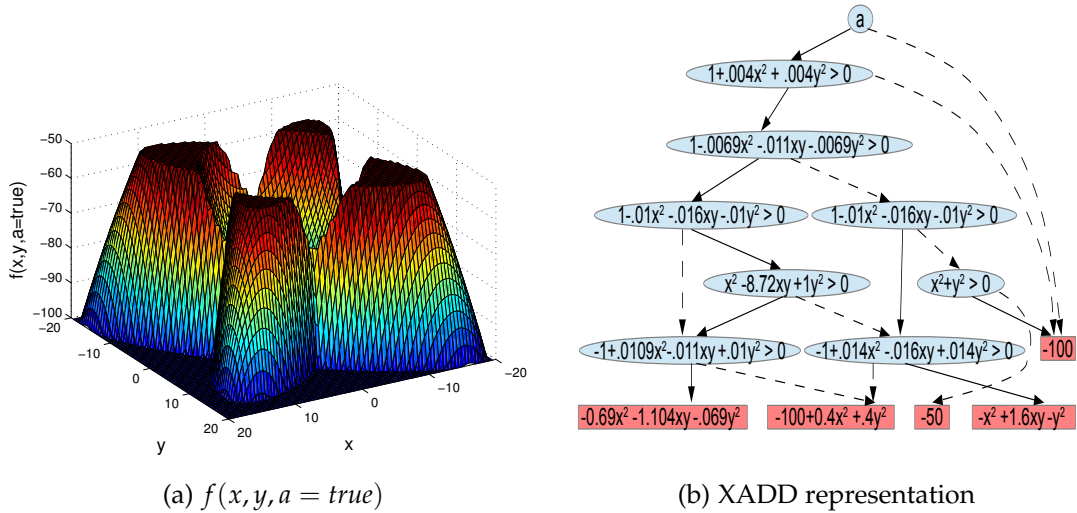


Figure 4.3: A plot of a function $f(x, y, a = \text{true})$ for continuous variables x and y and boolean variable a ; and its XADD representation.

4.3.3 XADD Operations

Here we review the symbolic operations required to perform SDP using the XADD data structure. Operations are conducted on *reduced* form XADDs, that is XADDs where inconsistent branches are removed. By conducting operations on reduced XADDs we are able to limit the number of XADD nodes required to represent a case statement. We categorise operations on XADDs as either unary or binary.

Unary operations on an XADD $\mathcal{X}$ include restriction, substitution and continuous integration. Restriction of a binary variable b in $\mathcal{X}$ to some logical formula ϕ is achieved by appending ϕ to each decision node in $\mathcal{X}$. Substitution into $\mathcal{X}$ is performed by applying a set of variables and their substitutions to each case statement in $\mathcal{X}$.

Binary operations between two XADDs $\mathcal{X}$ and $\mathcal{Y}$ include addition, subtraction, multiplication, max and min. All of the aforementioned operations are performed on *reduced* forms of $\mathcal{X}$ and $\mathcal{Y}$ via the Apply algorithm [Sanner et al., 2011].

4.4 Summary

Structured discrete and continuous solution techniques exploit the internal structure of MDPs and POMDPs. Internal structure in the state space $\mathcal{S}$ and, in the case of POMDPs, observation space $\mathcal{O}$ can be factored, which in turn can lead to more compact representations of the transition function $\mathcal{T}$, reward function $\mathcal{R}$ and, in the case of POMDPs, observation function $\mathcal{Z}$. Compact representations of the aforementioned functions are formalised as Dynamic Bayesian Networks, which are implemented as tree based data structures. Together, compact representations and efficient data structures mitigate the effect of the curse of dimensionality, which plagues MDPs and POMDPs in large state, and/or, observation spaces.

Symbolic dynamic programming (SDP) [Boutilier et al., 2001] is the process of performing dynamic programming via symbolic manipulations. In this chapter we presented case statements and case operations, which are the building blocks of SDP operations and algorithms. Case statements are capable of representing arbitrary piecewise functions. Each piecewise function is specified by an arbitrary logical combination of boolean variables and linear inequalities over discrete or continuous variables. A set of operations of unary and binary operations on case statements, that are pertinent to this thesis, were also presented. However, certain case operations, such as maximisation, lead to a result which is larger than its operands.

Extended Algebraic Decision Diagrams were introduced as an efficient and compact data structure capable of representing case statements whilst maintaining tractability. Operations on XADDs and their complications, such as the need to prune redundant nodes, were also presented.

In the subsequent chapters we utilise SDP and XADDs to calculate *exact* solutions to MDPs and POMDPs with continuous state spaces.

Closed-form solutions to Sequential Market Making with Inventory

Financial markets are well known for their volatility [Mandelbrot, 1963]. A prime driver of high volatility are periods of high uncertainty, such as those following price shocks, about the value of the assets being traded. These periods can lead to sparse trading, where there are few offers and counter-offers, which effectively stop markets from functioning. Market-makers (MMs) can be used in these situations to attract trading and maintain order in markets. Examples of markets that employ MMs include the NASDAQ, which allows for multiple market-makers to compete, and prediction markets [Hanson, 2003, Hanson, 2007].

Market-makers do not perform their services *ex gratia*. They aim to extract a profit, known as the bid-ask spread. When setting the *bid* and *ask* price, market-makers must trade-off between setting prices to extract maximum profit from the next trade, versus setting prices to get as much information about the new value of the asset so as to generate larger profits from future trades.

In this chapter we recast the sequential market making problem as a new general class of *continuous state* Partially Observable MDP (POMDP) [Kaelbling et al., 1998] that subsumes our proposed market making model and admits closed-form solutions. By doing so, we are the first to address the open question of sequentially optimal market making with inventory based reasoning by extending the seminal work Glosten and Milgrom [Glosten and Milgrom, 1985], Amihud and Mendelson [Amihud and Mendelson, 1980] and Das and Magdon-Ismael [Das and Magdon-Ismael, 2009]. We also present an SDP based algorithm that can efficiently derive a value function and policy for this class of POMDPs.

5.1 Optimal Sequential Market Making Model

The market making model used in this thesis extends the seminal theoretical model of Glosten and Milgrom [Glosten and Milgrom, 1985] and the sequential model of Das and Magdon-Ismael [Das and Magdon-Ismael, 2009] by incorporating the inven-

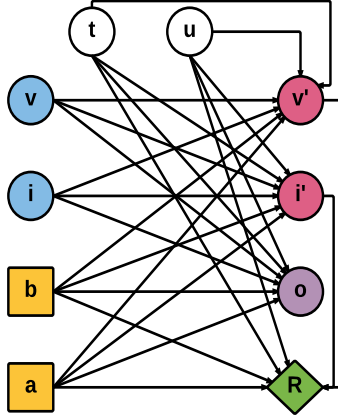


Figure 5.1: The optimal Sequential Market Making POMDP. The model consists of variables v - value, i - inventory, actions b - *bid*, a - *ask*, intermediate variables t and u , observation - o , and reward - R . Primed variables represent the “next state” of the variable. Arrows represent a dependency.

tory control conditions of Amihud and Mendelson [Amihud and Mendelson, 1980] as well as arbitrary belief state priors, trader models and asymmetric bid-ask spreads.

The aim of the MM is to maximise its reward over a planning horizon $\mathcal{H}$. In order to do this the MM must trade-off profit taking, which can be seen as exploiting a certain bid-ask pair, and price discovery, where the MM explores other bid-ask pairs in $\mathcal{A}$. Figure (5.1) shows our model graphically.

The POMDP formulation of optimal sequential market making presented in this section allows for the separation of belief state considerations from the dynamic programming solution. In subsequent sections we show that our model leads to an elegant piecewise linear structure in the α -functions derived through a symbolic dynamic programming solution. This is in stark contrast the approach of Das and Magdon-Ismail [Das and Magdon-Ismail, 2009] which required nonlinear operations that complicate the consideration of inventory.

We begin by formulating the MM’s sequential decision problem in the POMDP framework.

State space $\mathcal{S}$

$\mathcal{S} = \langle v, i \rangle$, where $v \in \mathbb{R}^+$ represents the value of the asset and $i \in \mathbb{N}$ represents the MM’s inventory.

Action space $\mathcal{A}$

$\mathcal{A} = \{(bid_1, ask_1), \dots, (bid_N, ask_N)\}$ represents a finite set of allowed N bid-ask pairs, where $bid_n, ask_n \in \mathbb{R}^+$ and $bid_n < ask_n$.

The *bid* represents the price at which the MM is willing to buy one unit of the asset and the *ask* is the price at which the MM is willing to sell one unit of the asset.

Observation space $\mathcal{O}$

$\mathcal{O} = \{buy, sell, hold\}$ represents the possible actions of a trader at each time step.

At each time step the MM interacts with a trader t drawn from a heterogeneous population of traders with a known prior distribution. The trader has an uninformed estimate of the value of the asset $v_t = v + \epsilon_t$, where $\epsilon_t \in \mathbb{R}$ specifies the noise.

We allow the traders to be specified according to two trader models: (i) Glosten Milgrom (GM); and (ii) Discrete noise (D). In the GM setting, traders are either informed or uninformed, with ϵ_t set to 0.0. The Discrete setting involves informed, over-valuing and under-valuing traders, with ϵ_t set to 0.0, constant $c \in \mathbb{R}$ and constant $-c \in \mathbb{R}$, respectively.

With the exception of uninformed traders, an arriving trader of type t will execute a *buy* order at an ask price ask_n if $v_t + \epsilon_t > ask_n$, a *sell* order at the bid price bid_n if $v_t + \epsilon_t < bid_n$, and will *hold* otherwise. Uninformed traders will *buy*, *sell* or *hold* with equal probability, irrespective of their valuation and the bid and ask prices. We use an intermediate variable $u \sim \mathbb{P}(\mathcal{O})$, with $\mathbb{P}(u) = 1/3$ for uninformed trader actions.

Transition function $\mathcal{T}$

The transition function $\mathcal{T}$ for each state variable in $\mathcal{S}$ is given by:

$$\begin{aligned} \mathcal{T}(i'|i, v, bid, ask, t = \text{informed}, u) &= \delta \left[i' - \begin{cases} v > ask + \epsilon_t \wedge (i \geq 1) : & i - 1 \\ v < bid + \epsilon_t : & i + 1 \\ \text{otherwise} : & i \end{cases} \right] \\ \mathcal{T}(i'|i, v, bid, ask, t = \text{uninformed}, u) &= \delta \left[i' - \begin{cases} u = \text{buy} \wedge (i \geq 1) : & i - 1 \\ u = \text{sell} : & i + 1 \\ \text{otherwise} : & i \end{cases} \right] \\ \mathcal{T}(v'|i, v, bid, ask, t, u) &= \delta[v' - v] \end{aligned}$$

The Dirac function $\delta[\cdot]$ ensures that the transitions are valid conditional probability functions that integrate to 1.0. The value v is assumed to be fixed, but unknown to the MM. The inventory i increments or decrements according to the observed trader action.

Observation function $\mathcal{Z}$

The observation function $\mathcal{Z}$ for each trader action is specified by:

$$\begin{aligned}
\mathcal{Z}(\text{buy}|i, v, \text{bid}, \text{ask}, t = \text{informed}, u) &= \begin{cases} v > \text{ask} + \epsilon_t \wedge (i \geq 1) : & 1 \\ \text{otherwise} : & 0 \end{cases} \\
\mathcal{Z}(\text{sell}|i, v, \text{bid}, \text{ask}, t = \text{informed}, u) &= \begin{cases} v < \text{bid} + \epsilon_t : & 1 \\ \text{otherwise} : & 0 \end{cases} \\
\mathcal{Z}(\text{hold}|i, v, \text{bid}, \text{ask}, t = \text{informed}, u) &= \begin{cases} (v > \text{bid} + \epsilon_t) \wedge (v < \text{ask} + \epsilon_t) : & 1 \\ \text{otherwise} : & 0 \end{cases} \\
\mathcal{Z}(o|i, v, \text{bid}, \text{ask}, t = \text{uninformed}, u) &= \mathbb{I}[o = u]
\end{aligned}$$

$\mathcal{Z}$ encodes the signal received by the MM upon the action taken by the trader. We note that information is conveyed only by the direction of the trade. In the case of an uninformed trader, the observation follows directly from the probabilistic choice determined in the transition function.

Reward function $\mathcal{R}$

The reward function $\mathcal{R}$, which constrains inventory to be non-negative, is specified as:

$$\begin{aligned}
\mathcal{R}(i', v', \text{bid}, \text{ask}, t = \text{informed}, u) &= \begin{cases} (v' + \epsilon_t > \text{ask}) \wedge (i' \geq 1) : & \text{ask} \\ (v' + \epsilon_t < \text{bid}) : & -\text{bid} \\ (v' + \epsilon_t > \text{bid}) \wedge (v' + \epsilon_t < \text{ask}) : & 0 \\ (v' + \epsilon_t > \text{ask}) \wedge (i' < 0) : & 0 \\ \text{otherwise} : & -\infty \end{cases} \\
\mathcal{R}(i', v', \text{bid}, \text{ask}, t = \text{uninformed}, u) &= \begin{cases} u = \text{buy} \wedge (i' \geq 1) : & \text{ask} \\ u = \text{sell} : & -\text{bid} \\ u = \text{hold} : & 0 \end{cases}
\end{aligned}$$

The reward received by the MM is dependent upon the action executed by the trader. In the case of a *buy* order, the MM receives the *ask* price, a *sell* order results in the MM paying the *bid* price whereas a *hold* order results in no loss or gain. Our market making model allows for an expressive class of reward functions that are piecewise linear in the value, inventory, and *bid* and *ask* prices. Hard constraints, such as the non-negative inventory condition, can be encoded using $-\infty$. Soft constraints, such as penalising linear deviations from an target inventory level, can be encoded using finite piecewise linear rewards.

5.1.1 Solution Techniques

Point-based Value Iteration (PBVI) [Pineau et al., 2003] mitigates the intractability of exact VI by considering only a finite subset of the belief space $\mathcal{B} = \{b_0, \dots, b_N\}$ during the Bellman backup operation. By restricting the belief space to $\mathcal{B}$, PBVI can only calculate approximate solutions. However, this apparent loss of optimality is mitigated by its ability to compute successful policies for much larger domains.

PBVI for POMDPs with discrete and finite $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$ can be written as follows:

$$\begin{aligned} \alpha_i^{a,o,h} &= \mathcal{R}(s, a) \cdot \frac{1}{|\mathcal{O}|} + \\ &\quad \gamma \cdot \sum_{s' \in \mathcal{S}} \Omega(o, a, s') \cdot \mathcal{T}(s, a, s') \cdot \alpha_i^{h-1}(s') \quad \forall \alpha_i^{h-1} \in V^{h-1} \\ \Gamma^{a,o,h} &= \bigcup_i \left\{ \alpha_i^{a,o,h} \right\} \\ \Gamma_b^{a,h} &= \sum_{o \in \mathcal{O}} \operatorname{argmax}_{\alpha \in \Gamma^{a,o,h}} (\alpha \cdot b) \end{aligned} \tag{5.1}$$

$$\begin{aligned} \alpha_b^h &= \operatorname{argmax}_{\Gamma_b^{a,h}, a \in \mathcal{A}} \left(\Gamma_b^{a,h} \cdot b \right), \quad \forall b \in \mathcal{B} \\ V^h &= \bigcup_{b \in \mathcal{B}} \left\{ \alpha_b^h \right\} \end{aligned} \tag{5.2}$$

In the formulation above, the initial value function V^0 is set as it was for VI. PBVI implicitly prunes dominated α -functions twice via the argmax operation in Equations (5.1) and (5.2), respectively. Thus, the final $|V^h|$ is limited to $|\mathcal{B}|$, which is in stark contrast to exact VI where $|V^h| = |\mathcal{A}| |V^{h-1}| |\mathcal{O}|$.

Standard PBVI can be extended to the continuous state setting by simply replacing s , s' and $\sum$ by their continuous state counterparts $\vec{x}$, $\vec{x}'$ and $\int$, respectively. The $\alpha \cdot b$ computation in Equations (5.1) and (5.2) is also replaced by its continuous state counterpart, $\int (\alpha \cdot b) d\vec{x}$. Despite the ease of mathematically formulating the algorithm for continuous states, PBVI cannot be easily implemented due to the infinite number of states in $\vec{x}$. In the next section we show that a large subclass of continuous state POMDPs can be solved optimally for arbitrary horizons using Symbolic Dynamic Programming.

5.2 SDP for Continuous State POMDPs

In this section we make our second contribution by showing how the POMDP based optimal sequential market making model can be solved in closed-form via a Symbolic Dynamic Programming (SDP) [Boutilier et al., 2001] based version of Point Based Value Iteration (PBVI) [Pineau et al., 2003].

Symbolic PBVI for continuous $\mathcal{S}$ and discrete $\mathcal{A}$ POMDPs can be written solely

Algorithm 6: SDP-PBVI(POMDP, $\mathcal{H}$, $\mathcal{B}$) $\longrightarrow V^h$

```

1 begin
2    $h := 0, V^h := \{0\}$ 
3   while  $h < \mathcal{H}$  do
4      $h := h + 1, V^h := \emptyset$ 
5     foreach  $b \in \mathcal{B}$  do
6       foreach  $a \in \mathcal{A}$  do
7          $\Gamma_b^{a,h} := \emptyset, \alpha_b^{a,h} := 0$ 
8         foreach  $o \in \mathcal{O}$  do
9            $\Gamma^{a,o,h} := \emptyset$ 
10          foreach  $\alpha_i^{h-1} \in V^{h-1}$  do
11             $\alpha_i^{a,o,h} := \text{see Equation (5.4)}$ 
12             $\Gamma^{a,o,h} := \Gamma^{a,o,h} \cup \{\alpha_i^{a,o,h}\}$ 
13             $\alpha_b^{a,o} := \arg\max_{\alpha \in \Gamma^{a,o,h}} (\alpha \cdot b)$ 
14             $\alpha_b^{a,h} := \alpha_b^{a,h} + \alpha_b^{a,o}$ 
15             $\Gamma_b^{a,h} := \Gamma_b^{a,h} \cup \{\alpha_b^{a,h}\}$ 
16           $\alpha_b^h := \arg\max_{a \in \mathcal{A}} (\Gamma_b^{a,h} \cdot b)$ 
17           $V^h := V^h \cup \{\alpha_b^h\}$ 
18          // Terminate if early convergence;
19          if  $V^h = V^{h-1}$  then
20            break
21  return  $V^h$ 

```

in terms of the following case operations:

$$\begin{aligned}
\alpha_i^{a,o,h} &= \mathcal{R}(\vec{x}, a) \cdot \frac{1}{|\mathcal{O}|} + \\
&\quad \gamma \otimes \int \Omega(o, a, \vec{x}') \otimes \mathcal{T}(\vec{x}, a, \vec{x}') \otimes \alpha_i(\vec{x}') d\vec{x}', \quad \forall \alpha_i(\vec{x}') \in V^{h-1} \\
\Gamma^{a,o,h} &= \bigcup_i \{\alpha_i^{a,o,h}\} \\
\Gamma_b^{a,h} &= \sum_{o \in \mathcal{O}} \arg\max_{\alpha \in \Gamma^{a,o,h}} (\alpha \cdot b) \\
\alpha_b^h &= \arg\max_{\Gamma_b^{a,h}, a \in \mathcal{A}} (\Gamma_b^{a,h} \cdot b), \quad \forall b \in \mathcal{B} \\
V^h &= \bigcup_{b \in \mathcal{B}} \{\alpha_b^h\}
\end{aligned} \tag{5.3}$$

Algorithm (6) presents SDP-PBVI, a symbolic PBVI algorithm that calculates closed-form solutions for the optimal market making model presented in Section 5.1. The algorithm calculates the optimal h -stage-to-go value function via the operations de-

defined above.

To calculate the optimal h -stage-to-go value function we modify the Bellman backup in Equation (5.3) to the following form:

$$\alpha_i^{a,o,h} = \bigoplus_u \bigoplus_t \mathbb{P}(t) \cdot \mathcal{R}(\vec{x}, a, t, u) \cdot \frac{1}{|\mathcal{O}|} + \gamma \otimes \int \Omega(o, a, \vec{x}', t, u) \otimes \mathcal{T}(\vec{x}, a, \vec{x}', t, u) \otimes \alpha_i(\vec{x}') d\vec{x}', \quad \forall \alpha_i(\vec{x}') \in V^{h-1} \quad (5.4)$$

In Equation (5.4) the Backup operation is calculated as an expectation over trader types t in a given trader model and observations $u \in \{\text{buy}, \text{sell}, \text{hold}\}$ from uninformed traders. This Symbolic PBVI algorithm extends the approach of [Zamani et al., 2012a] to the more general case of rewards with piecewise linear conditions and linear values.

We remark that all of the SDP operations used in the Symbolic PBVI algorithm are closed-form for LPWC representations of $\Omega(\cdot)$ and LPWL representations of $\mathcal{T}(\cdot)$ and $\mathcal{R}(\cdot)$ [Sanner et al., 2011, Zamani et al., 2012a]. A LPWC $\Omega(\cdot)$ ensures that the α -functions are LPWL and closed-form after the Bellman backup operation; the integration operation in Equation (5.4) results in a LPWL function. In contrast, a LPWL $\Omega(\cdot)$ would not result in a closed-form LPWL α -function. Therefore, by induction, the Symbolic PBVI value function V^h remains closed-form for arbitrary horizons for the optimal sequential market making model.

Symbolic PBVI gives a lower bound on the optimal exact Value Iteration [Monahan, 1982] value function and, if all belief states are enumerated to $\mathcal{H}$, the solution is optimal. The lower bound policy also guarantees that the MM makes at least as much profit as indicated by the PBVI value function evaluated at a belief state.

5.3 Experimental Results

In this section we investigate the properties of our novel sequential market making model and demonstrate the following attributes: (i) sequential optimality; (ii) inventory sensitivity; (iii) the importance of flexible trader models in calculating sequentially optimal policies; and (iv) the computational tractability of Symbolic PBVI for market making with inventory.

For each of the analyses an initial set of beliefs over $\mathcal{S} = \langle v, i \rangle$ is used to generate the set of all reachable beliefs $\mathcal{B}$ to $\mathcal{H}$. Trader proportions under the Glosten Milgrom and Discrete models were set to $\mathbb{P}(t) = 0.5$ and $\mathbb{P}(t) = 1/3$, respectively. The noise under the Discrete model for over-valuing and under-valuing traders was set to 10.0 and -10.0 , respectively. While we can effectively solve for $\mathcal{H} = 30$ or more, we intentionally restrict the horizon to $\mathcal{H} = 3$ in the initial set of evaluations for purposes of interpretation and explanation.

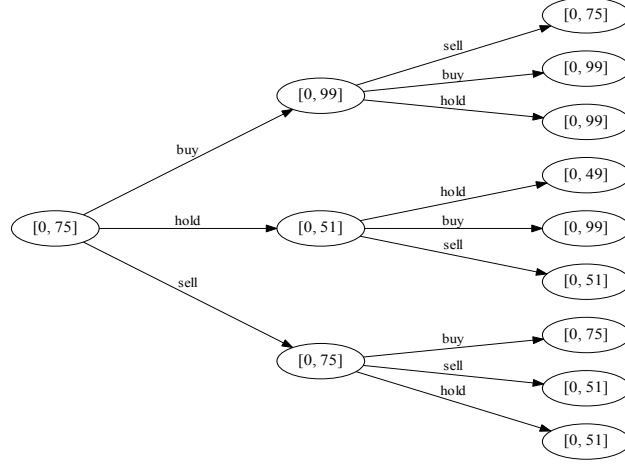


Figure 5.2: Sequentially optimal $\mathcal{H} = 3$ policy. Nodes represent actions $a \in \mathcal{A}$ and arrows represent an observation $o \in \mathcal{O}$.

5.3.1 Sequential Optimality

In Figure (5.2) we present the sequentially optimal policy of our MM operating within the Glosten Milgrom trader model with an initial belief of $\langle \{\mathcal{U}[0.0, 50.0], \mathcal{U}[50.0, 100.0]\}, 1.0 \rangle$ and $\mathcal{A} = \{(0.0, 25.0), (0.0, 49.0), (0.0, 51.0), (0.0, 75.0), (0.0, 99.0)\}$. The MM changes the optimal $\mathcal{H} = 3$ bid-ask pair of $[0.0, 75.0]$ in response to trader actions: the *ask* price is raised in response to a *buy*, lowered following a *hold* and unchanged in the event of a *sell* order. The actions chosen at $\mathcal{H} = 2$ reflect the optimal choice given the MM's updated belief. At $\mathcal{H} = 1$ the MM chooses actions that ensures that a trader will *buy*, contingent upon having positive inventory. The sequence of actions chosen by the MM demonstrates the way in which it chooses *bid* and *ask* prices to learn from the order stream, whilst also bounding inventory.

5.3.2 Inventory Sensitivity

In Figure (5.3) we present the optimal $\mathcal{H} = 3$ actions of our MM operating within the Glosten Milgrom trading model with different initial beliefs and $\mathcal{A} = \{(bid, ask) | (bid < ask), bid, ask \in \{20.0, 40.0, 60.0, 80.0, 100\}\}$. Upon comparing Figure (5.3a) and Figure (5.3b) we notice that, with the exception of the $i = 0.0$ case, a narrow initial value belief has a narrower optimal bid-ask spread. It is also evident that the MM uses higher *ask* prices when the trader population comprises of a higher proportion of uninformed traders. In summary, from Figure (5.3) we can clearly see that the MM sets its actions based on its initial beliefs, characteristics of the trader population and inventory levels.

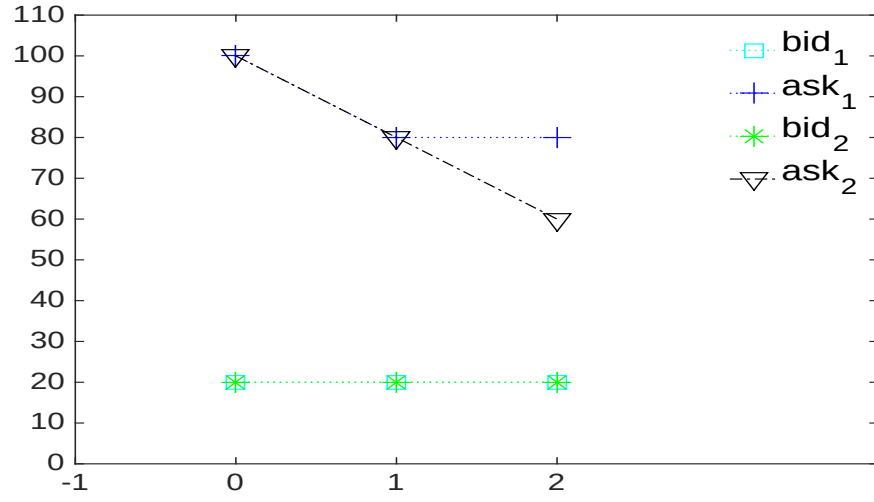
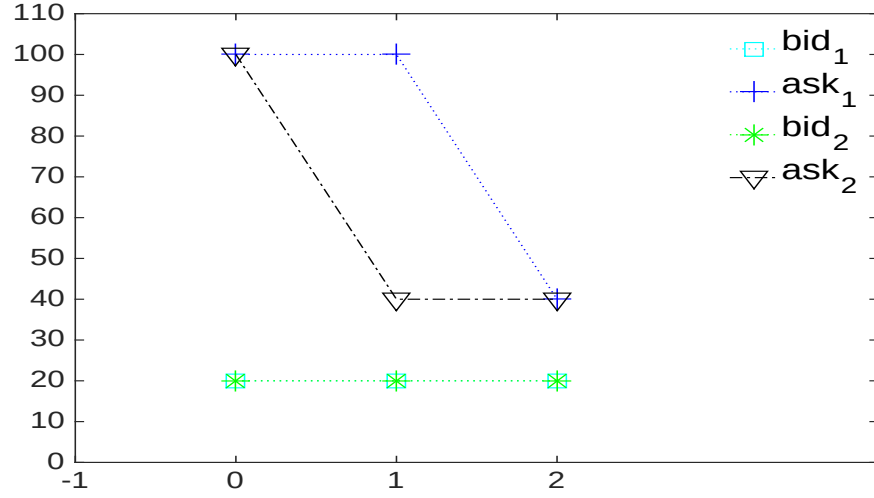
(a) Initial value belief of $v \sim \mathcal{U}[0.0, 100.0]$.(b) Initial value belief of $v \sim \mathcal{U}[49.0, 51.0]$.

Figure 5.3: Optimal $\mathcal{H} = 3$ bid and ask prices under different market settings. The initial inventory level $i \in \{0, 1, 2\}$. (bid_1, ask_1) and (bid_2, ask_2) are the optimal prices for a trader population comprising of 50% and 10% uninformed traders, respectively.

5.3.3 Trader Models

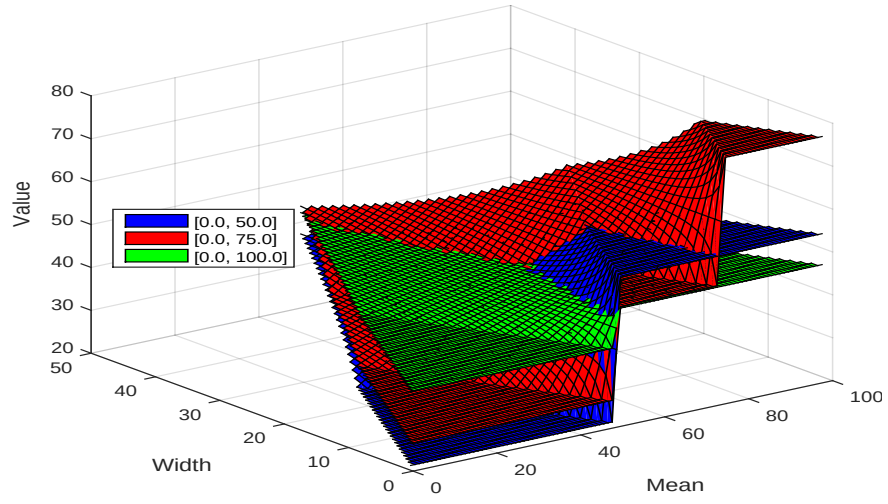
In the previous two experiments we have shown how the optimal sequential market making model presented in this thesis can be used to calculate sequentially optimal and inventory sensitive policies, the two major contributions to algorithmic market making. In this experiment we demonstrate the flexibility of the model by examining the affect of different trader models on the MM's optimal *bid* and *ask* prices. In Figure (5.4) we present the optimal α -functions of our MM operating within the Glosen Milgrom and Discrete trading model with an initial belief of $\langle \{\mathcal{U}[0.0, 50.0], \mathcal{U}[50.0, 100.0]\}, 1.0 \rangle$ and $\mathcal{A} = \{(bid, ask) | (bid < ask), bid \in \{0.0, 25.0, 50.0, 75.0\}, ask \in \{50.0, 75.0, 100.0\}\}$.

In Figure (5.4a) we can observe three distinct optimal actions in the α -function plots. If we set the width to 0.0 and vary the mean, we notice that $[0.0, 100.0]$ exploits the actions of uninformed traders who are equally likely to *sell* or *buy*. In the regions where $[0.0, 50.0]$ and $[0.0, 75.0]$ are optimal, we note that as the mean increases above the ask price, the MM is more likely to receive *buy* orders from an informed trader. In Figure (5.4b) we can see two interesting trends with respect to the optimal actions. Firstly, when the MM is certain about the value, they select actions with increasing *ask* prices. This phenomenon also occurs when uncertainty about the mean value increases. For example, when the MM becomes increasingly uncertain about a mean value of 70.0, they use actions with a wider bid-ask spread. From Figure (5.4) it is evident that models of trader noise have a dramatic affect on the MM's optimal actions. Therefore, it is critical for an algorithmic market making model to be flexible with regards to its reasoning about traders.

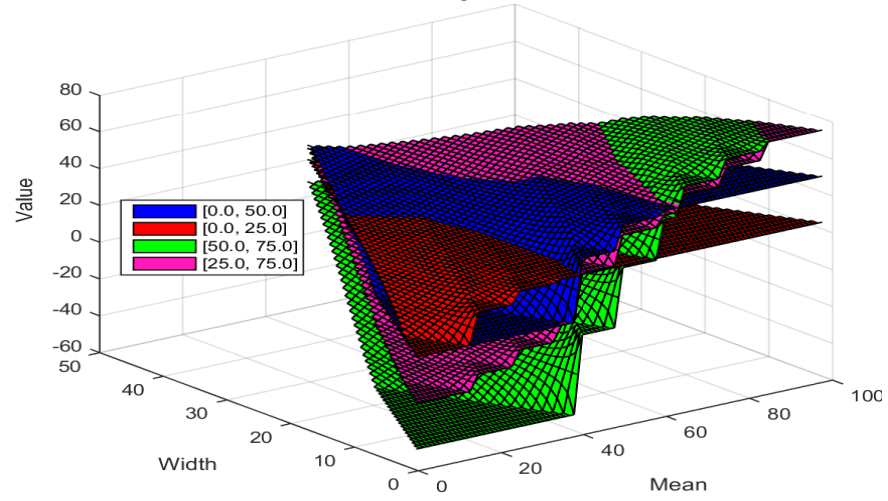
5.4 Related Work

In recent history, research into algorithmic market making has focussed on financial [Das, 2008] and prediction market settings [Chen and Pennock, 2007, Othman and Sandholm, 2012]. Market-making in prediction markets starts from the notion of a proper Market Scoring Rule (MSR) [Hanson, 2003]. Proper MSRs are both myopically incentive compatible, that is, they incentivise agents to move prices to their valuation, and are also loss-bounded, at least in bounded pay-off markets. Despite these attractive features, proper MSRs are, in general, loss-making [Pennock and Sami, 2007] and MMs using proper MSRs must be subsidised in order to perform their task of liquidity provision. Although there has been some theoretical work on the design of market makers that can adapt their spread over time [Othman et al., 2010] they can be highly sensitive to the price path [Brahma et al., 2012].

Learning based approaches to market making in financial markets attempt to infer either underlying values or future prices based on trader models [Das, 2008, Das and Magdon-Ismail, 2009], or time-series analysis of prices [Chakraborty and Kearns, 2011]. Such models have been used to formulate efficient algorithms under zero-profit (competitive) conditions where, in perfect competition, the MM is pushed to setting *bid* and *ask* prices that yield a zero expected profit [Das, 2005]. An algo-



(a) Glosten-Milgrom model.



(b) Discrete model.

Figure 5.4: Optimal $\mathcal{H} = 3$ α -functions for $i = 1.0$ under different trader models. The upper surface of the plot indicates the optimal α -function for the market-maker's value belief defined by $v \sim \mathcal{U}[m - w, m + w]$, for a given mean m and width w .

rithm for a profit-maximising (monopolist) MM was presented by Das and Magdon-Ismail [Das and Magdon-Ismail, 2009]. In this setting, *bid* and *ask* prices are set in a way that maximises the total discounted profit obtained by the MM. The profit-maximising market making algorithm was cast as a belief state MDP where the MM’s belief about the value of the asset was described by a Gaussian density function. The MM set *bid* and *ask* prices based on the solution of the Bellman equation for a Gaussian belief state approximation of this MDP. While these models are promising in terms of their potential to provide liquidity while simultaneously operating at a profit, they come with no guarantees on worst case loss, since they do not take inventory into account when setting prices.

This work has built on PBVI methods [Pineau et al., 2003] for solving large POMDPs by approximating the value function over a set of representative reachable belief points. While PBVI has been well explored for the discrete state case including symbolic extensions using decision diagrams [Poupart, 2005], much less work has focused on the continuous state case. Three notable continuous state extensions of PBVI include: (i) the seminal work by Porta *et al* [Porta et al., 2006] which extends PBVI to the case of value functions which admit an exact mixture of Gaussians representation; (ii) a POMDP formalisation used for Bayesian reinforcement learning [Poupart et al., 2006] which examined the case of value functions which admit a mixture of Dirichlets representation; and (iii) a Symbolic Dynamic Programming approach to solving POMDPs with continuous observations but with highly restricted piecewise constant reward and belief states [Zamani et al., 2012b]. In this thesis, we present a more general SDP solution to a class of discrete observation POMDPs with more general piecewise linear reward transitions, piecewise constant observations, and arbitrary belief distributions. As far as we are aware this work is the first closed-form PBVI solution for this expressive class of POMDPs.

5.5 Summary

In this chapter we have introduced the first optimal solution to sequential market making with inventory by using a new subclass of *continuous state* POMDPs. We have also presented a novel symbolic dynamic programming implementation of the PBVI algorithm that can calculate exact closed-form computations of the PBVI Bellman backups. Our contributions enable the calculation of sequentially optimal policies with bounded loss under a variety of market models and conditions.

Closed-form solutions to a Subclass of Continuous Stochastic Games

Modelling competitive sequential interactions between agents has important applications within economic and financial decision-making. Stochastic games [Shapley, 1953] provide a convenient framework to model sequential interactions between non-cooperative agents. In zero-sum stochastic games, participating agents have diametrically opposing goals. A reinforcement learning solution to discrete state zero-sum stochastic games was presented by Littman [Littman, 1994]. Closed-form solutions for the continuous state case remain unknown, despite the general importance of this formalism — zero-sum continuous state stochastic games provide a convenient framework with which to model robust sequential optimisation in adversarial settings including domains such as option valuation on derivative markets. The difficulty of solving zero-sum continuous state stochastic games originates from the need to calculate a Nash equilibrium for every state, of which there are infinitely many.

In this chapter we characterise a subclass of zero-sum continuous state stochastic games with restricted reward and transition functions that can be solved exactly via parameterised linear optimisation. Furthermore, we provide an algorithm that solves this subclass of stochastic games exactly and optimally using Symbolic Dynamic Programming (SDP) [Boutilier et al., 2001, Sanner et al., 2011, Zamani et al., 2012a] for arbitrary horizons. We demonstrate the efficacy of our novel framework and solution technique by calculating the first known exact solutions to a variety of zero-sum continuous stochastic games. Of particular note is the first known exact solution to a binary option stochastic game between a trader and an adversarial market.

6.1 Zero-sum Continuous Stochastic Games

Continuous state stochastic games (CSGs) are formally defined by the tuple $\langle \mathcal{S}, \mathcal{A}_1, \dots, \mathcal{A}_n, \mathcal{T}, \mathcal{R}_1, \dots, \mathcal{R}_n \rangle$. We now describe each unique element of the tuple in turn.

State space $\mathcal{S}$

$\mathcal{S}$ specifies the state space $(\vec{x}) = (x_1, \dots, x_n)$, where each $x_j \in \mathbb{R}$ ($1 \leq j \leq n$) is continuous.

Action space $\mathcal{A}$

$\mathcal{A}_i$ is the action set available to agent i , for $i \in [1, n]$.

Transition function $\mathcal{T}$

$\mathcal{T}$ is a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \dots \times \mathcal{A}_n \rightarrow \Delta(\mathcal{S})$, where $\Delta(\mathcal{S})$ is the set of probability distributions over the state space $\mathcal{S}$.

Reward function $\mathcal{R}$

The reward function $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_1 \times \dots \times \mathcal{A}_n \rightarrow \mathbb{R}$, encodes the individual preferences of agent i , for $i \in [1, n]$. Zero-sum CSGs are a type of CSG involving two agents with diametrically opposing goals. Under these conditions the reward structure for the game can be represented by a single reward function since an agent's reward function is simply the opposite of their opponent's. The objective of each agent is to maximise its expected discounted future rewards in the minimax sense. That is, each agent views its opponent as acting to minimise the agent's reward.

6.1.1 Solution Techniques

In general, agent i 's objective is to find a policy, $\pi_i : \mathcal{S} \rightarrow \sigma_i(\mathcal{A}_i)$, which maximises the expected sum of discounted rewards over horizon $\mathcal{H}$. Here $\sigma_i(\mathcal{A}_i)$ specifies probability distributions over action set $\mathcal{A}_i$. The optimal policy in a CSG may be stochastic, that is, it may define a mixed strategy for each state.

The optimal solution to zero-sum CSGs can be calculated via the following recursive functions:

$$Q^h(\vec{x}, a_1, a_2) = \mathcal{R}(\vec{x}, a_1, a_2) + \gamma \cdot \int \mathcal{T}(\vec{x}, a_1, a_2, \vec{x}') \cdot V^{h-1}(\vec{x}') d\vec{x}' \quad (6.1)$$

$$V^h(\vec{x}) = \max_{m \in \sigma(\mathcal{A}_1)} \min_{a_2 \in \mathcal{A}_2} \sum_{a_1 \in \mathcal{A}_1} Q^h(\vec{x}, a_1, a_2) \cdot m_{a_1} \quad (6.2)$$

Together Equations (6.1) and (6.2) define a recursive method to calculate the optimal solution to zero-sum CSGs. The policy for the opponent can be calculated by applying symmetric reasoning and the Minimax theorem [Neumann, 1928]. Zero-sum CSGs can be solved using a technique analogous to that presented in [Littman, 1994]. Namely, the value function in Equation (6.2) can be reformulated as the following continuous optimisation problem:

$$\begin{aligned}
& \text{maximise } V^h(\vec{x}) \\
& \text{subject to} \\
& V^h(\vec{x}) \leq \sum_{a_1 \in \mathcal{A}_1} Q^h(\vec{x}, a_1, a_2) \cdot m_{a_1} \quad \forall a_2 \in \mathcal{A}_2 \\
& \sum_{a_1 \in \mathcal{A}_1} m_{a_1} = 1; \quad m_{a_1} \geq 0 \quad \forall a_1 \in \mathcal{A}_1
\end{aligned} \tag{6.3}$$

This optimisation problem cannot be easily solved using existing techniques due to two factors: (i) there are infinitely many states in $\vec{x}$; and (ii) constraint (6.3) is non-linear in $\vec{x}$ and m_{a_1} for general representations of $Q^h(\vec{x}, a_1, a_2)$. To further illustrate the second limitation, consider $Q^h(\vec{x}, a_1, a_2)$ in the form of a linear function in x , for some a_1 and a_2 :

$$Q^h(\vec{x}, a_1, a_2) = \sum_j c_j \cdot x_j \tag{6.4}$$

Substituting Equation (6.4) into constraint (6.3) yields:

$$V^h(\vec{x}) \leq \sum_{a_1 \in \mathcal{A}_1} m_{a_1} \sum_j c_j \cdot x_j \quad \forall a_2 \in \mathcal{A}_2. \tag{6.5}$$

It is clear from Equation (6.5) that a linear representation of $Q^h(\vec{x}, a_1, a_2)$ leads to a nonlinear constraint, where m_{a_1} must be optimal with respect to the free variable $\vec{x}$. This results in a parameterised nonlinear program, whose optimal solutions are known to be NP-hard [Bennett and Mangasarian, 1993, Petrik and Zilberstein, 2011].

At this point we present the first key insight of this chapter: we can transform constraint (6.3) from a parameterised nonlinear constraint to a piecewise linear constraint by imposing the following restrictions: (i) restricting the reward function, $\mathcal{R}(\vec{x}, a_1, a_2)$, to piecewise constant functions; and (ii) restricting the transition function, $\mathcal{T}(\vec{x}, a_1, a_2, \vec{x}')$, to piecewise linear functions. As a result, $V^h(\vec{x})$ and $Q^h(\vec{x}, a_1, a_2)$ will be piecewise constant functions, thereby guaranteeing a tractable solution to constraint (6.3).

One key challenge still remains, namely, dealing with the infinitely many states in $\vec{x}$. We know that the $V^h(\vec{x})$ and $Q^h(\vec{x}, a_1, a_2)$ functions have structure, but are unable to derive them. Furthermore, given known structures for $V^h(\vec{x})$ and $Q^h(\vec{x}, a_1, a_2)$ we must determine the restrictions that guarantee a tractable solution. The SDP framework in conjunction with its closed-form operations provide answers to both of these concerns.

In the next section we show that zero-sum CSGs, with the aforementioned restrictions, can be solved optimally for arbitrary horizons using Symbolic Dynamic Programming.

6.2 SDP for Zero-sum Continuous Stochastic Games

In this section we will show that a subclass of zero-sum continuous stochastic games with (a) piecewise constant rewards; and (b) piecewise linear transition functions can be solved exactly and in closed-form by using SDP.

To calculate the exact solution to zero-sum CSGs we begin by replacing all functions and operations in Equations (6.1) and (6.2) by their case statement equivalents. That is, we exchange operations such as $+$, $\times$ and $\min$, by their symbolic equivalents, $\oplus$, $\otimes$ and casemin , respectively, and express $\mathcal{R}(\vec{x}, a_1, a_2)$, $\mathcal{T}(\vec{x}, a_1, a_2, \vec{x}')$, $V^0(\vec{x})$ as case statements. m_{a_1} is also encoded as a trivial case statement representing an uninstantiated symbolic variable:

$$m_{a_1} = \left\{ \top : m_{a_1} \right.$$

The optimal solution to zero-sum CSGs can now be described by the following recursive SDP equations:

$$Q^h(\vec{x}, a_1, a_2) = \mathcal{R}(\vec{x}, a_1, a_2) \oplus \gamma \otimes \int \mathcal{T}(\vec{x}, a_1, a_2, \vec{x}') \otimes V^{h-1}(\vec{x}') d\vec{x}' \quad (6.6)$$

$$\tilde{Q}^h(\vec{x}, a_2) = \sum_{a_1 \in \mathcal{A}_1} Q^h(\vec{x}, a_1, a_2) \otimes m_{a_1} \quad (6.7)$$

$$V^h(\vec{x}) = \max_m \text{casemin} \left(\text{casemin}_{a_2 \in \mathcal{A}_2} \left(\tilde{Q}^h(\vec{x}, a_2) \right), \mathbb{I} \right) \quad (6.8)$$

Equation (6.7) calculates a symbolic Q function weighted by the variable m_{a_1} for each a_1 . In Equation (6.8) the inner casemin operation is calculated with respect to $\tilde{Q}^h(\vec{x}, a_2)$ instantiated with a particular a_2 . The *indicator* function $\mathbb{I}$ serves as the summation constraint $\sum_{a_1 \in \mathcal{A}_1} m_{a_1} = 1$ and ensures that the subsequent $\max$ operation returns legal values for the m_{a_1} . The indicator is defined as follows:

$$\mathbb{I} = \begin{cases} \forall a_1 \in \mathcal{A}_1 \quad [(m_{a_1} \geq 0) \wedge (m_{a_1} \leq 1) \wedge (\sum m_{a_1} = 1)] : +\infty \\ \forall a_1 \in \mathcal{A}_1 \neg [(m_{a_1} \geq 0) \wedge (m_{a_1} \leq 1) \wedge (\sum m_{a_1} = 1)] : -\infty \end{cases}$$

The function $\mathbb{I}$ returns $+\infty$ when the conjunction of all constraints on each m_{a_1} are satisfied and $-\infty$, otherwise.

In Algorithm 7 we present CSG-VI, a methodology to calculate the optimal h -stage-to-go value function through Equations (6.6) to (6.8). In the algorithm we notationally specify the arguments of the Q^h and V^h functions when they are instantiated by an operation. For the base case of $h = 0$, we set $V^0(\vec{x})$, expressed in case statement form, to zero or to the terminal reward. For all $h > 0$ we apply the previously defined SDP operations in the following steps:

1. Prime the Value Function. In line (6) we set up a substitution $\theta =$

$\{x_1/x'_1, \dots, x_m/x'_m\}$, and obtain $V^{h'} = V^h\theta$, the “next state”.

2. Discount and add Rewards. We assume that the reward function contains primed variables and incorporate it in line (8).
3. Continuous Regression. For the continuous state variables in $\vec{x}$ lines (10) – (11) follow the rules of integration w.r.t. a δ function [Sanner et al., 2011]. This yields the following symbolic substitution: $\int f(x'_j) \otimes \delta[x'_j - g(\vec{z})] dx'_j = f(x'_j) \{x'_j/g(\vec{z})\}$, where $g(\vec{z})$ is a case statement and $\vec{z}$ does not contain x'_j . The latter operation indicates that any occurrence of x'_j in $f(x'_j)$ is symbolically substituted with the case statement $g(\vec{z})$.
4. Incorporate Agent 1’s strategy. At this point we have calculated $Q^h(\vec{x}, a_1, a_2)$, as shown in Equation (6.6). In lines (13) - (14) we weight the strategy for a specific a_1 by m_{a_1} . We note that m_{a_1} is a case statement representing an uninstantiated symbolic variable.
5. Case Minimisation. In lines (16) – (17) we compute the best case for a_2 as a function of all other variables, as shown in Equation (6.7).
6. Incorporate Constraints. In line (19) we incorporate constraints on the m_{a_1} variable: $\sum_{a_1 \in \mathcal{A}_1} m_{a_1} = 1$ and $m_{a_1} \geq 0 \wedge m_{a_1} \leq 1 \quad \forall a_1 \in \mathcal{A}_1$. The casemin operator ensures that all case partitions which involve illegal values of m_{a_1} are mapped to $-\infty$.
7. Maximisation. In lines (22) – (23) we compute the best response strategy for every state. We note that this can only be done via symbolic methods since there are infinitely many states. At this point we have calculated $V^h(\vec{x})$ as shown in Equation (6.8).

It can be proved that all of the SDP operations used in Algorithm 1 are closed form for LPWC or LPWL functions [Sanner et al., 2011, Zamani et al., 2012a]. Given a LPWC $V^0(\vec{x})$ and that SDP operations are closed form, the resulting $V^{h+1}(\vec{x})$ is also LPWC. Therefore, by induction $V^{h+1}(\vec{x})$ is LPWC for all horizons h .

In the next section we demonstrate how SDP can be used to compute exact solutions to a variety of zero-sum continuous stochastic games.

6.3 Experimental Results

In this section we evaluate our novel SDP solution technique for zero-sum CSGs on three continuous domains: (i) continuous stochastic matching pennies; (ii) binary option stochastic game; and (iii) robust energy production. The results represent the first known exact solutions to these domains.

Algorithm 7: $\text{CSG-VI}(\text{CSG}, \mathcal{H}, \mathbb{I}) \rightarrow (V^h)$

```

1 begin
2    $V^0 := 0, h := 0$ 
3   while  $h < \mathcal{H}$  do
4      $h := h + 1$ 
5     // Prime the value function
6      $Q^h := \text{Prime}(V^{h-1})$ 
7     // Discount and add Rewards
8      $Q^h := \mathcal{R}(\vec{x}, a_1, a_2, \vec{x}') \oplus (\gamma \otimes Q^h)$ 
9     // Continuous Regression
10    for all  $x'_j \in Q^h$  do
11       $Q^h := \int Q^h \otimes T(x'_j | a_1, a_2, \vec{x}) d_{x'_j}$ 
12    // Incorporate Agent 1's strategy
13    for all  $a_1 \in \mathcal{A}_1$  do
14       $Q^h := Q^h \oplus (Q^h(a_1) \otimes \{\top : m_{a_1}\})$ 
15    // Case Minimisation
16    for all  $a_2 \in \mathcal{A}_2$  do
17       $Q^h := \text{casemin}(Q^h, Q^h(a_2))$ 
18    // Incorporate constraints
19     $Q^h := \text{casemin}(Q^h, \mathbb{I})$ 
20    // Maximisation
21     $V^h = Q^h$ 
22    for all  $a_1 \in \mathcal{A}_1$  do
23       $V^h := \max_{m_{a_1}} V^h(m_{a_1})$ 
24    // Terminate if early convergence
25    if  $V^h = V^{h-1}$  then
26      break
27  return  $(V^h)$ 

```

6.3.1 Continuous Stochastic Matching Pennies

Matching pennies is a well known zero-sum game with a mixed strategy Nash Equilibrium [Osborne, 2004]. In this thesis we extend the standard formulation of the game by incorporating continuous state and sequential decisions, while still maintaining the zero-sum nature of the reward.

We define continuous stochastic matching pennies as an extensive form game between two players $p \in \{1, 2\}$. The aim of a player is to maximise its expected discounted pay-off at a fixed horizon $\mathcal{H}$. Our game is played within the interval $[0, 1]$, two fixed variables $c \in [0, 1)$ and $d \in (0, 1]$ with $(c < d)$, are used to partition the interval into three regions $r \in \{1, 2, 3\}$. Each region is associated with its own zero-sum reward structure. The continuous state variable $x \in [0, 1]$ is used to specify that region the players are competing within.

At each horizon $(h \leq H)$ each player executes an action $a_p \in \{\text{heads}_p, \text{tails}_p\}$.

Player 1 “wins” if both players choose the same action. Otherwise, Player 2 wins. The joint actions of the players affect the state x as follows:

$$\mathbb{P}(x'|x, a_1, a_2) = \delta \left[x' - \begin{cases} (heads_1) \wedge (heads_2) \wedge (x \geq k) : & x - k \\ (heads_1) \wedge (tails_2) \wedge (x \leq 1) : & x + k \\ (tails_1) \wedge (heads_2) \wedge (x \geq k) : & x + k \\ (tails_1) \wedge (tails_2) \wedge (x \leq 1) : & x - k \end{cases} \right]$$

The constant $k \in (0, 1)$ is a step size that perturbs the state x . If Player 1 wins, the state moves to the left by k , otherwise it moves to the right by k . The Dirac function $\delta[\cdot]$ ensures that the transitions are valid conditional probability functions that integrate to 1.

We define the rewards obtained by Player 1 in region r as:

$$\mathcal{R}_1^r = \begin{cases} (heads_1) \wedge (heads_2) : & \alpha_1^r \\ (heads_1) \wedge (tails_2) : & \alpha_2^r \\ (tails_1) \wedge (heads_2) : & \alpha_3^r \\ (tails_1) \wedge (tails_2) : & \alpha_4^r \end{cases}$$

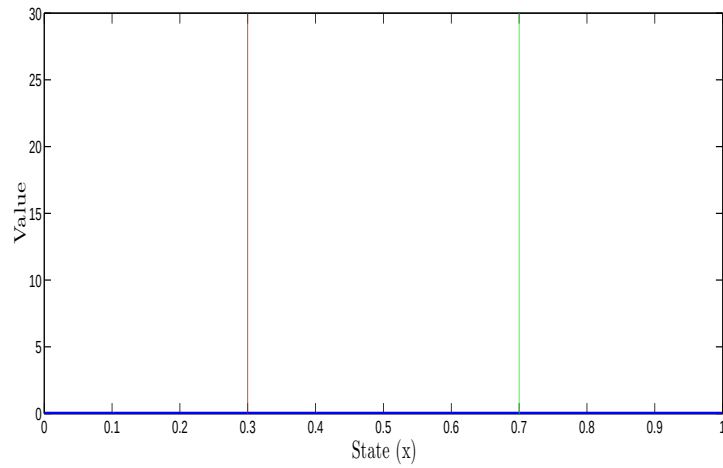
Here we restrict $\alpha_i^r \in \mathbb{R}$. The rewards obtained by Player 2 in the same region are simply $-\mathcal{R}_1^r$. Given this reward formulation we specify two different reward structures: symmetric and asymmetric. In a symmetric reward structure $\alpha_1^r = \alpha_4^r$ and $\alpha_2^r = \alpha_3^r$. An example of this reward structure is shown in Table 6.1. Under a symmetric reward setting the expected reward for each player is the same across all regions r . In an asymmetric reward structure we allow each of the α_i^r to differ in both sign and magnitude. Table 6.2 shows an example of an asymmetric reward structure. Under an asymmetric setting the expected reward for each player may vary across each region r . This gives a player an incentive to reach regions with a higher expected reward.

Table 6.1: Symmetric reward structure for Player 1.

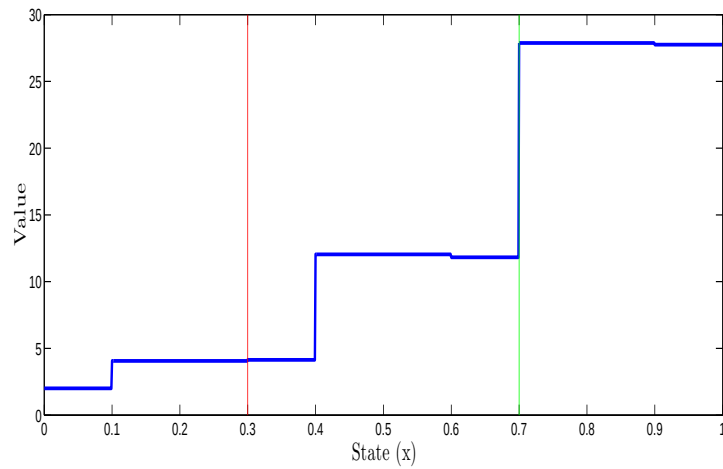
	Region 1	Region 2	Region 3
$(heads_1) \wedge (heads_2)$	10	5	20
$(heads_1) \wedge (tails_2)$	-10	-5	-20
$(tails_1) \wedge (heads_2)$	-10	-5	-20
$(tails_1) \wedge (tails_2)$	10	5	20

We investigate the continuous stochastic matching pennies game under both symmetric and asymmetric reward structures. For both experiments the threshold values are set to $c = 0.3$ and $d = 0.7$. The step size is $k = 0.3$.

Figure (6.1a) shows the results of the continuous stochastic matching pennies



(a) Symmetric rewards.



(b) Asymmetric rewards.

Figure 6.1: The optimal value functions of the continuous stochastic matching pennies game for Player 1 at horizon 4 under (a) symmetric and (b) asymmetric reward structures. Threshold values are set to $c = 0.3$ and $d = 0.7$ and are highlighted in red and green, respectively. The step size is $k = 0.3$.

Table 6.2: Asymmetric reward structure for Player 1.

	Region 1	Region 2	Region 3
$(heads_1) \wedge (heads_2)$	1	5	7
$(heads_1) \wedge (tails_2)$	-3	-5	-2
$(tails_1) \wedge (heads_2)$	0	-5	10
$(tails_1) \wedge (tails_2)$	2	5	20

game using the symmetric reward structure given in Table 6.1. The results show that the expected reward for Player 1 remains at zero over all 4 horizons, irrespective of the state x . The symmetric reward structure clearly shows that both players achieve the same expected reward in all regions r . This in turn ensures that both players are indifferent between their pure strategies. Hence, the expected reward for each player is zero in all regions. This result corresponds to the well known solution of the matching pennies game with symmetric rewards and serves as a proof of concept for our novel solution technique.

Figure (6.1b) shows the effect of the asymmetric reward structure given in Table 6.2. The figure shows that Player 1 achieves the highest expected reward in Region 3, followed by Region 2 and finally by Region 1. This corresponds to the expected reward within each region of Table 6.2. The results indicate that the Player 1 is no longer indifferent between its pure strategies in each region and may take short-term losses to reach more favourable regions.

6.3.2 Binary Option Stochastic Game

Binary options are financial instruments that allow an investor to bet on the outcome of a yes/no proposition. The proposition typically relates to whether the price of a particular asset that underlies the option will rise above or fall below a specified amount, known as the strike price, $\kappa \in \mathbb{R}$. When the option reaches maturity the investor receives a fixed pay-off if their bet was correct and nothing otherwise.

We frame the valuation of a binary option as an extensive form zero-sum game between a trader and the market. The aim of the trader is to maximise their expected discounted pay-off at a fixed horizon $\mathcal{H}$ through buying and selling options within an adversarial market. The problem has two state variables: the underlying market value of the asset $v \in [0, 100]$ and the trader's inventory of options $i \in \mathbb{N}$.

At each time step the trader can execute one of three actions $a_{trd} \in \{buy_{trd}, sell_{trd}, hold_{trd}\}$, where buy_{trd} refers to a request to buy an option from the market, $sell_{trd}$ refers to a request to sell an option to the market and $hold_{trd}$ is equivalent to taking no action. The market can execute one of two actions: $a_{mkt} \in \{sell_{mkt}, nsell_{mkt}\}$, where $sell_{mkt}$ corresponds to selling an option to the trader and $nsell_{mkt}$ corresponds to not selling to the trader.

The joint actions of the trader and market, a_{trd} and a_{mkt} , respectively, affect both the underlying market value of the asset and the trader's inventory. For the sake of simplicity we assume that the market value may increase or decrease by fixed step

sizes, $u \in \mathbb{R}$ for an increase and $d \in \mathbb{R}$ for a decrease.

The trader's option inventory dynamics are given by:

$$\mathbb{P}(i'|v, i, a_{trd}, a_{mkt}) = \delta \left[i' - \begin{cases} (buy_{trd}) \wedge (sell_{mkt}) : & i + 1 \\ (sell_{trd}) \wedge (i > 0) : & i - 1 \\ otherwise : & i \end{cases} \right]$$

It should be noted that under this formulation the market will always buy an option from the trader when the trader selects $sell_{trd}$. The market value changes according to:

$$\mathbb{P}(v'|v, i, a_{trd}, a_{mkt}) = \delta \left[v' - \begin{cases} (buy_{trd}) \wedge (sell_{mkt}) : & v + u \\ (sell_{trd}) \wedge (i > 0) : & v - d \\ otherwise : & v \end{cases} \right]$$

Assuming that the strike price $\kappa \in [0, 100]$, the rewards obtained by the trader are given by:

$$\mathcal{R}_{trader} = \begin{cases} (sell_{trd}) \wedge (i > 0) \wedge (v > \kappa) : & 1 \\ otherwise : & 0 \end{cases}$$

The market's reward is simply the additive inverse of the trader's reward. Hence, the binary option game is zero-sum.

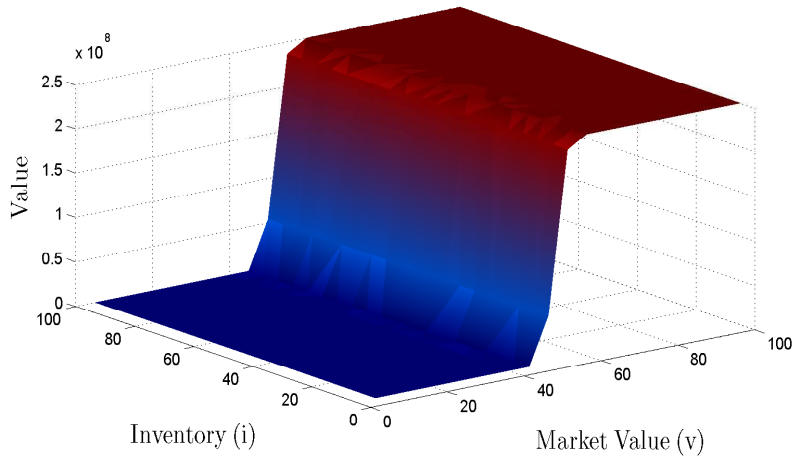


Figure 6.2: The optimal value function of the binary option stochastic game for the trader at horizon 20. The strike price is set to $\kappa = 45.0$ and the increment and decrement values are set to $u = 1.0$ and $d = 1.0$, respectively. Under the domain specification the value function is invariant to the inventory i .

In Figure (6.2) we show the optimal value function for the binary option game at horizon 20. The strike price is set to $\kappa = 45.0$ and the increment and decrement values, u and d are both set to 1.0. The value function clearly shows that under this formulation the trader achieves the most reward by selling options as soon $v > \kappa$. Selling an option causes the underlying value to decrease. Once the value falls beneath the strike price, the trader will buy options, which increases the underlying value. This leads to the continual cycling of buying and selling of the option at values close to the strike price κ . In essence the trader behaves like a market maker in that they take both sides of the transaction at values near κ . We note that while Figure (6.2) is invariant to the inventory of options i , its inclusion is critical for the correct formalisation of the domain.

6.3.3 Robust Energy Production

The provision of energy resources is an integral component of any economy. Energy providers must be able to produce energy in response to changes in energy demand. In situations where demand exceeds supply, an energy crisis may occur. In this thesis we investigate energy production from the viewpoint of an energy provider responsible for supplying energy in an adversarial environment.

We define our energy production domain as an extensive form zero-sum game between an energy provider and nature. The aim of the energy provider is to maximise its expected discounted reward at a fixed horizon H by changing production levels in response to changes in demand. The domain has two state variables: the production level $p \in \mathbb{R}^+$ and the energy demand $d \in \mathbb{R}^+$.

At each time step the energy provider can execute one of two actions $a_{prd} \in \{inc_{prd}, dec_{prd}\}$, where inc_{prd} refers to increasing energy production and dec_{prd} refers to decreasing energy production. Nature can also execute one of two actions $a_{nat} \in \{inc_{dem}, dec_{dem}\}$, where inc_{dem} refers to increasing energy demand and dec_{dem} refers to decreasing energy demand. We specify the increase in the amount of energy produced or demanded by $prd_u, nat_u \in \mathbb{R}^+$ and a corresponding decrease by $prd_d, nat_d \in \mathbb{R}^+$.

The joint actions of the energy provider and nature, a_{prd} and a_{nat} , respectively, affect the production level as follows:

$$\mathbb{P}(p'|d, p, a_{prd}, a_{nat}) = \delta \left[p' - \begin{cases} (inc_{prd}) : & p + prd_u \\ (dec_{prd}) \wedge (p > prd_d) : & p - prd_d \\ otherwise : & p \end{cases} \right]$$

The energy demand changes according to:

$$\mathbb{P}(d'|d, p, a_{prd}, a_{nat}) = \delta \left[d' - \begin{cases} (inc_{dem}) : & d + nat_u \\ (dec_{dem}) \wedge (d > nat_d) : & d - nat_d \\ otherwise : & d \end{cases} \right]$$

The reward obtained by the energy provider are specified as:

$$\mathcal{R}_{prd} = \begin{cases} (p < d) : & -100 \\ otherwise : & 0 \end{cases}$$

We note that under this reward structure, failure to meet energy demand is heavily penalised, whereas meeting or even exceeding demand are given the same reward. Nature's reward is simply the additive inverse of the energy provider's reward.

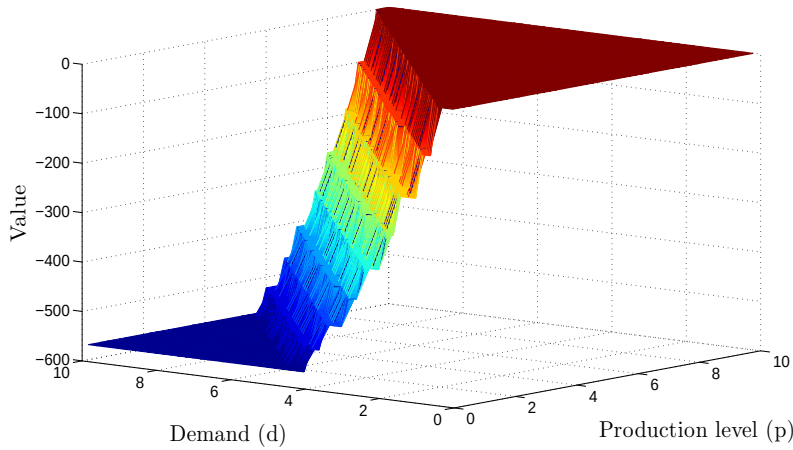


Figure 6.3: The optimal value function of the robust energy production game for the producer at horizon 8. The production and demand increase and decrease variables were set to $prd_u = prd_d = 1.0$ and $nat_u = nat_d = 0.5$, respectively.

In Figure (6.3) we show the optimal value function for the robust energy production game at horizon 8. The production and demand increase and decrease variables were set to $prd_u = prd_d = 1.0$ and $nat_u = nat_d = 0.5$, respectively. The value function shows that the energy provider achieves the highest value when the energy provided meets or exceeds demand. The value function is lowest when the demand exceeds supply, which is in accordance with the reward structure. The value function clearly decreases in a step-wise manner from the point where the production level meets demand, indicating that production levels just beneath demand have a higher value than those well below demand.

6.4 Related Work

Solutions to stochastic games have been proposed from within both game theory and reinforcement learning. The first algorithm, game theoretic or otherwise, for finding a solution to a stochastic game was given by Shapley [Shapley, 1953]. The algorithm repeatedly calculates a value function $V(s)$ over discrete states which converges to an optimal value function $V^*(s)$, which represents the expected discounted future reward if both players in the game followed the game’s Nash equilibrium. Shapley’s algorithm is in essence an extension of the value iteration algorithm to stochastic games. A reinforcement learning based solution to stochastic games was first introduced by Littman [Littman, 1994]. Littman’s algorithm, Minimax-Q, extends the traditional Q-learning algorithm for MDPs to zero-sum discrete stochastic games. The algorithm converges to the stochastic game’s equilibrium solution. Hu and Wellman [Hu and Wellman, 1998] extended Minimax-Q to general-sum games and proved that it converges to a Nash equilibrium under certain restrictive conditions. Although both reinforcement learning based algorithms are able to calculate equilibrium solutions they are limited to discrete state formulations of stochastic games. In this thesis we calculate exact solutions to continuous state formulations of stochastic games, under certain restrictions. In this thesis we provide the first known exact closed-form solution to a subclass of continuous state zero-sum stochastic games defined by a piecewise constant reward and piecewise linear transition.

Several techniques have been put forward to tackle continuous state spaces in MDPs. Li and Littman [Li and Littman, 2005] describe a method to approximate intermediate piecewise linear value functions by piecewise constant functions, which results in approximate solutions to continuous state MDPs. Furthermore, Li and Littman only allow for rectilinearly aligned constraints in their reward and transition functions, not arbitrary linear constraints, and cannot handle general linear transition models without approximating. Our SDP method provides exact solutions without these restrictions, which makes SDP strictly more general. Finally, Li and Littman did not consider game-theoretic extensions of their work or the parameterised optimisation problem that these extensions entail.

Symbolic dynamic programming techniques have been previously used to calculate exact solutions to single agent MDPs with both continuous state and actions in a variety of non-game theoretic domains [Sanner et al., 2011, Zamani et al., 2012a]. In this thesis we present the first application of SDP to stochastic games with concurrently acting agents.

6.5 Summary

In this chapter we have characterised a subclass of zero-sum continuous stochastic games that can be solved exactly via parameterised linear optimisation. We have also presented a novel symbolic dynamic programming algorithm that can be used to calculate exact solutions to this subclass of games for arbitrary horizons. The algorithm was used to calculate the first known exact solutions to zero-sum continuous

stochastic game formulations of binary option valuation, robust energy production and matching pennies.

Analytic Decision Analysis for Parameterised Hybrid MDPs

Traditional MDP solution techniques, such as those presented in Chapter 3 often assume that the parameters of the model are known. However, in practice, model parameters are usually estimated from limited data or elicited from humans and hence are naturally uncertain. It is often critical in real world applications to: (i) perform inverse learning of parameters of multi-objective rewards; (ii) perform sensitivity analyses of policies to various parameter settings; and (iii) analyse and optimize policy performance as a function of policy parameters. Formalizing models to address each of the aforementioned use cases is often fraught, due to the specification leading to hybrid (mixed discrete and continuous state and/or action) MDPs with nonlinear and/or piecewise structure that have been traditionally very difficult to solve.

In this chapter we present the unifying framework of *Parameterised Hybrid MDPs* (PHMDPs), which enables the inverse learning of parameters of multi-objective rewards, parameter sensitivity analysis and non-convex analysis and optimization of policy parameters. We also introduce an algorithm that solves this class of PHMDPs exactly and in closed-form by defining a parameterised variant of Symbolic Dynamic Programming (SDP) [Boutilier et al., 2001] extended to hybrid MDPs [Sanner et al., 2011]. By leveraging SDP we are able to define nonlinear symbolic functions that can be exactly optimized by state-of-the-art non-convex optimizers such as dReal [Gao et al., 2013]. We demonstrate the efficacy of the PHMDP framework and novel solution technique by calculating the first known exact solutions to a variety of nonlinear sequential decision problems. Of particular interest is the first known exact sensitivity analysis of portfolio transaction models over the full range of model parameters.

7.1 Parameterised Hybrid MDPs

A parameterised hybrid Markov Decision Process (PHMDP) is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{D}, \Theta \rangle$. We now describe each unique element of the tuple in turn; the remaining components of the tuple are as defined in Section 2.1.

State space $\mathcal{S}$

$\mathcal{S}$ specifies the state space $(\vec{d}, \vec{x}) = (d_1, \dots, d_m, x_1, \dots, x_n)$, where each $d_i \in \{0, 1\}$ ($1 \leq i \leq m$) is discrete and each $x_j \in \mathbb{R}$ ($1 \leq j \leq n$) is continuous.

Action space $\mathcal{A}$

$\mathcal{A}_s^t$ specifies a finite set of state and horizon dependent actions.

Transition function $\mathcal{T}$

PHMDPs are naturally factored [Boutilier et al., 1999] in terms of the state variables $\vec{d}$ and $\vec{x}$. Hence, the joint transition model can be written as:

$$\mathcal{T} : \mathbb{P} \left(\vec{d}', \vec{x}' \mid \vec{d}, \vec{x}, a, \vec{\theta} \right) = \prod_{i=1}^m \mathbb{P} \left(d'_i \mid \vec{d}, \vec{x}, a, \vec{\theta} \right) \prod_{j=1}^n \mathbb{P} \left(x'_j \mid \vec{d}, \vec{d}', \vec{x}, a, \vec{\theta} \right), \quad (7.1)$$

where $a \in \mathcal{A}_s^t$. The transition model permits discrete noise in the sense that $\mathbb{P} \left(x'_j \mid \vec{d}, \vec{d}', \vec{x}, a, \vec{\theta} \right)$ may condition on $\vec{d}'$, which are stochastically sampled according to their conditional probability functions [Zamani et al., 2012a, Sanner et al., 2011].

Reward function $\mathcal{R}$

$\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \vec{\theta} \rightarrow \mathbb{R}$ is the reward function, which encodes the preferences of the agent.

Parameter space Θ

$\vec{\theta} \in \Theta$ are free parameters from the parameter space Θ .

In the proceeding sections we demonstrate how the PHMDP framework can be specialized into models capable of: (i) investigating multi-objective reward criteria; (ii) exact parameter sensitivity analysis; and (iii) optimization of continuous non-convex policy parameters.

7.1.1 Inverse Learning for Multi-objective PHMDPs

The PHMDP framework can be specialized into a multi-objective setting by specifying the reward as $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^d$, where d is the dimension of the reward vector. In this formulation the parameter $\vec{\theta}^d$ specifies the linear scalarisation over the reward components (i.e., $\langle \mathcal{R}, \vec{\theta}^d \rangle$) and the value function can be expressed as a function of all possible scalarisations. Multi-objective PHMDPs can also be used for the inverse learning of the parameters of the multi-objective reward that explains observed behaviour.

7.1.2 Sensitivity Analysis for PHMDPs

To conduct a sensitivity analysis, we assume that $\vec{\theta}^d$ are unknown parameters and we solve the value function in Equation (7.3) as a parametric function, $V^{\pi^*}(\vec{d}, \vec{x}; \vec{\theta}^d)$, of these parameters. The parametric value function can then be displayed or analysed to allow a user to explore trade-offs between parameters $\vec{\theta}^d$ or the optimal policy π^* as a function of other state variables $(\vec{d}, \vec{x})$.

7.1.3 Nonlinear Parameterised Policy Optimization Methods for PHMDPs

PHMDP policy parameters can be analysed and optimized by parameterising a policy π by action parameters $\vec{\theta}^d$, and evaluating the value function with respect to the parameterised policy. The resulting non-convex policy value, as a function of $\vec{\theta}^d$, can be analysed and optimized using the parametric form of $V^{\pi}(\vec{d}, \vec{x}; \vec{\theta}^d)$ and all symbolic derivatives, up to any order, of this value function.

7.1.4 Solution Techniques

The Value Iteration algorithm presented in Algorithm (3) can be modified to solve PHMDPs. $V^{\pi^*}(\vec{d}, \vec{x}; \vec{\theta})$ and $Q^{\pi^*}(\vec{d}, \vec{x}, a; \vec{\theta})$ can be approximated by the following recursive relationship:

$$Q^h(\vec{d}, \vec{x}, a; \vec{\theta}) = \mathcal{R}(\vec{d}, \vec{x}, a; \vec{\theta}) + \gamma \cdot \sum_{\vec{d}'} \int_{\vec{x}'} \mathbb{P}(\vec{d}', \vec{x}' | \vec{d}, \vec{x}, a; \vec{\theta}) \cdot V^{h-1}(\vec{d}', \vec{x}'; \vec{\theta}) d\vec{x}' \quad (7.2)$$

$$V^h(\vec{d}, \vec{x}; \vec{\theta}) = \max_{a \in \mathcal{A}} \left\{ Q^h(\vec{d}, \vec{x}, a; \vec{\theta}) \right\} \quad (7.3)$$

$\mathbb{P}(\vec{d}', \vec{x}' | \vec{d}, \vec{x}, a; \vec{\theta})$ is specified in Equation (7.1).

Despite the ease with which VI can be modified to solve PHMDPs, two key challenges still remain: (i) dealing with infinitely many states in $\vec{x}$ and $\vec{\theta}$; and (ii) determining restrictions on $Q^h(\vec{d}, \vec{x}, a; \vec{\theta})$ and $V^h(\vec{d}, \vec{x}; \vec{\theta})$ that guarantee tractable solutions. In the next section we show that a parameterised extension of symbolic dynamic programming can be used to address these concerns and solve PHMDPs, optimally and in closed-form, for arbitrary horizons.

7.2 SDP for Parameterised Hybrid MDPs

To calculate the exact solution to PHMDPs we begin by replacing all functions and operations in Equations (7.2) and (7.3) by their case statement equivalents. The optimal solution to PHMDPs can now be described by the following recursive SDP

equations:

$$Q^h(\vec{d}, \vec{x}, a; \vec{\theta}) = \mathcal{R}(\vec{d}, \vec{x}, a; \vec{\theta}) \oplus \gamma \cdot \bigoplus_{\vec{d}'} \int_{\vec{x}'} \mathbb{P}(\vec{d}', \vec{x}' | \vec{d}, \vec{x}, a; \vec{\theta}) \otimes V^{h-1}(\vec{d}', \vec{x}'; \vec{\theta}) d\vec{x}' \quad (7.4)$$

$$V^h(\vec{d}, \vec{x}; \vec{\theta}) = \text{casemax}_{a \in \mathcal{A}} \{Q^h(\vec{d}, \vec{x}, a; \vec{\theta})\} \quad (7.5)$$

We note that all parameters θ_i that are free variables are encoded as $\delta[(\theta'_i - \theta_i)]$, indicating that they are stationary and hence do not change during the backup operation. Continuous state parameters $\vec{x}$ are handled in a similar fashion. All operations including action maximization will automatically condition the value on these parameters, yielding the parameterised value function in Equation (7.5).

In the case of discrete $\mathcal{A}$ it can be proved that all of the SDP operations used in Equations (7.4) and (7.5) are closed form for NPWN functions [Sanner et al., 2011]. In the case of continuous $\mathcal{A}$ all of the operations are closed form for only LPWC or LPWL functions [Zamani et al., 2012a].

7.2.1 Inverse Learning for Multi-objective PHMDPs

A possible formulation for the inverse learning problem for multi-objective MDPs is to constrain the Q-values corresponding to the observed behaviour and maximize $\vec{\theta}$, which can be interpreted as multi-objective weights that best explain the observed behaviour:

$$\max_{\vec{\theta}} \max_{a_k, a_{-k} \neq \pi} Q^t(\vec{d}, x, a_k; \vec{\theta}) \ominus Q^h(\vec{d}, x, a_{-k}; \vec{\theta}) \quad (7.6)$$

where x can either be fixed or a region specified in the constraints, a_k refers to the action taken under the policy π in a particular state and a_{-k} refers to all other available actions in that state.

PHMDPs with multi-objective $\mathcal{R}$ and linear scalarisation functions can be solved exactly and in closed-form by restricting the $\mathcal{R}$ to LPWC functions and $\mathcal{T}$ to LPWL functions. Given a LPWL $V^0(\vec{d}, \vec{x}; \vec{\theta})$ and that SDP operations are closed form, the resulting $V^1(\vec{d}, \vec{x}; \vec{\theta})$ is also LPWL. Therefore, by induction $V^{h+1}(\vec{d}, \vec{x}; \vec{\theta})$ is LPWL for all horizons $h \in \mathcal{H}$. Multi-objective PHMDPs with a nonlinear scalarisation function and NPWN $\mathcal{R}$ and $\mathcal{T}$ functions lead NPWN solutions, which are exact and closed-form [Sanner et al., 2011].

7.2.2 Sensitivity Analysis for PHMDPs

Sensitivity analysis for PHMDPs can be analysed exactly and in closed-form via SDP by first calculating Equation (7.5) and then taking symbolic derivatives, up to any order, with respect to the parameter $\vec{\theta}^d$.

7.2.3 Nonlinear Parameterised Policy Optimization Methods for PHMDPs

Parameterised policies $\pi(\vec{\theta}^d)$, where $\vec{\theta}^d$ may be nonlinear, for PHMDPs can be analysed exactly and in closed-form via SDP by substituting $\pi(\vec{\theta}^d)$ in for a in Equation (7.4). This precludes the need for action maximization in Equation (7.5). Because this function is parametric, it is possible to take symbolic derivatives up to any order i.e. $\nabla_{\vec{\theta}^d} Q^h(\vec{d}, \vec{x}, a; \vec{\theta}^d)$ and apply non-convex optimization tools, such as dReal or dOp [Gao et al., 2013], that exploit parametric knowledge of the function. Symbolic derivatives avoid the stochasticity issues associated with traditional policy gradient methods.

In the next section we demonstrate how PHMDPs can be combined with SDP and state-of-the-art nonlinear optimization techniques to solve hard nonlinear sequential decision problems that were previously impossible.

7.3 Experimental Results

In this section we demonstrate the efficacy of our novel framework by calculating the first known optimal solutions to the following difficult nonlinear sequential decision problems: (i) inverse learning of parameters in a multi-objective navigation domain; (ii) non-convex optimization of vaccination policy parameters in a public health setting; and (iii) exact sensitivity analysis of portfolio transaction models over the full range of model parameters. Furthermore, we show that our framework is scalable in both time and space. We use the nonlinear solver `fmincon` [The MathWorks Inc., 2010] in the experiments.

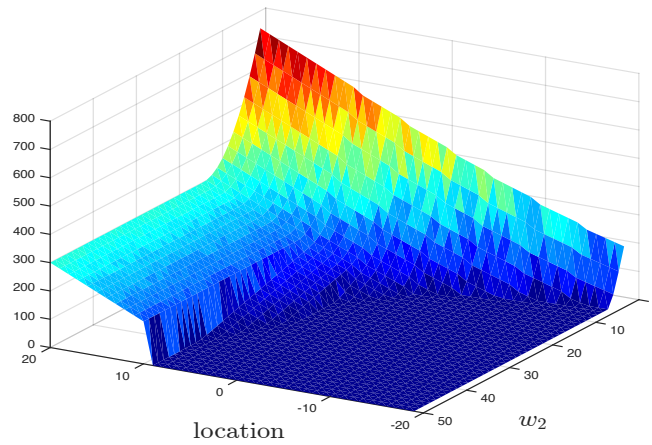
7.3.1 Inverse Learning for Multi-objective Navigation

In this domain we consider an autonomous vehicle moving in one dimension towards a goal region. At each stage the vehicle faces a trade-off between reaching the goal region and incurring a movement cost. The domain is specified as follows: $\mathcal{S} = \langle loc \rangle$, where loc is the location of the vehicle. $\mathcal{A} \in \{0.0, 5.0\}$ is the amount by which vehicle moves relative to its current location. $\mathcal{T}(loc' | loc, a) = \delta[loc' + (loc + a)]$, where $a \in \mathcal{A}$. $\mathcal{R}(\vec{w}, loc, loc') = w_1 \cdot \mathcal{R}_{\text{region}} + w_2 \cdot \mathcal{R}_{\text{move}}$ where,

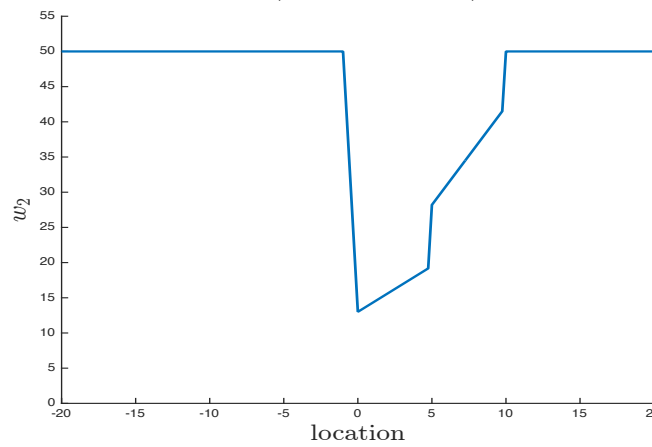
$$\mathcal{R}_{\text{region}}(loc') = \begin{cases} (loc' \geq 10.0) : & loc' \\ \text{otherwise} : & 0.0 \end{cases}$$

$$\mathcal{R}_{\text{move}}(loc, loc') = -(loc' - loc)$$

Figure (7.1a) shows the optimal value function at $\mathcal{H} = 15$. The function reveals that the magnitude of the movement cost the vehicle is willing to incur is inversely proportional to its distance from the goal region. In Figure (7.1b) we utilise techniques from inverse reinforcement learning [Ng and Russell, 2000] to learn the parameters (weights) of the multi-objective reward under a sub-optimal policy π of the



(a) $V^{\pi^*}(loc, w_2; w_1 = 1.0)$.



(b) Maximal $w_2 \in [0.0, 50.0]$ that explains a sub-optimal policy.

Figure 7.1: Value function and Nonlinear optimization for the Multi-objective Navigation domain.

form: $\pi(0 < loc < 10) = 5.0, \pi(loc < 0 \text{ or } loc > 10) = 0.0$. We note that w_2 was at its maximum allowable value when the vehicle did not move and that it was sufficiently low when the vehicle does move. As per the Figure (7.1a), we see that the robot is willing to incur a higher w_2 the closer it is to the goal region.

7.3.2 Influenza Public Health Policy

In this domain we address the optimization of a static parameterised vaccination policy under a model of Influenza epidemiology where the decision maker must balance the cost of vaccination and the burden of disease on the population. The domain is specified as follows: $\mathcal{S} = \langle s, i, r \rangle$, where s , i , and r refer to the size of the susceptible, infected and recovered sub-populations, respectively. $\mathcal{A} \in \{\pi(\nu)\}$ where $\nu \in [0.0, 1.0]$ is the proportion of s to vaccinate at each stage. The transition function $\mathcal{T}$ for each state variable in $\mathcal{S}$ is given by:

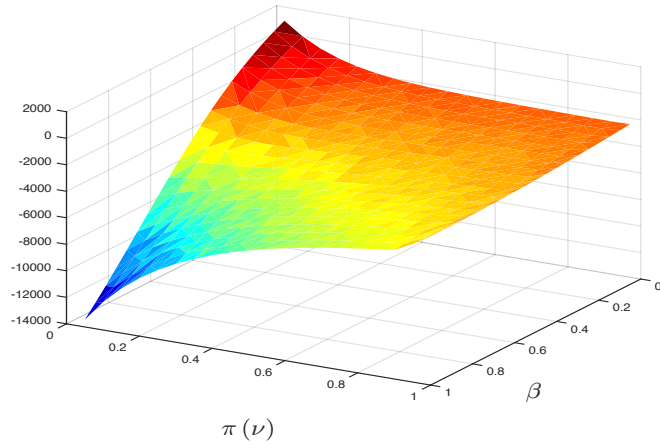
$$\begin{aligned}\mathcal{T}(s'|s, i, r, \pi(\nu)) &= \delta [s' - (s - \beta \cdot s \cdot i - \pi(\nu) \cdot s)] \\ \mathcal{T}(i'|s, i, r, \pi(\nu)) &= \delta [i' - (i + \beta \cdot s \cdot i - \lambda \cdot i)] \\ \mathcal{T}(r'|s, i, r, \pi(\nu)) &= \delta [r' - (r + \lambda \cdot i + \pi(\nu) \cdot s)]\end{aligned}$$

where β is the infection rate and λ is the spontaneous recovery rate. The reward is specified as $\mathcal{R}(cost_{\text{inf}}, cost_{\text{vaccine}}, s, i, r, \pi(\nu)) = (s \cdot (-cost_{\text{vaccine}} \cdot \pi(\nu) + (1 - \pi(\nu)))) - cost_{\text{inf}} \cdot i + r$. $cost_{\text{inf}}$ is the incident cost of infection, akin to a burden of disease, and $cost_{\text{vaccine}}$ is the unit cost of vaccination. We assume that the total population is constant and that vaccinated individuals go straight from s to r without being infected.

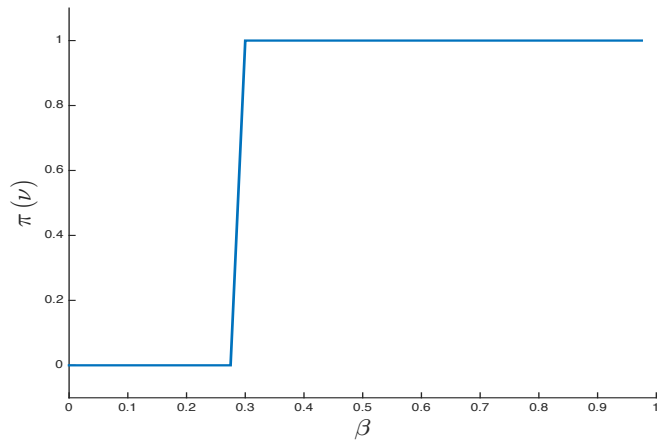
Figure (7.2a) shows the optimal value function at $\mathcal{H} = 7$ when $s = 1000.0, i = 100.0, r = 0.0, \lambda = 0.25, cost_{\text{vaccine}} = 4.0$ and $cost_{\text{inf}} = 10.0$. The value function shows that it is not always optimal to vaccinate the entire population. In fact, the non-convex optimization of the value function with respect to the policy parameter ν , shown in Figure (7.2b), reveals that it is optimal to vaccinate the entire population only when the infection rate $\beta > 0.25$. When $\beta > 0.25$, the *basic reproductive ratio* $R_0 (= \beta/\lambda)$ [Heffernan et al., 2005] exceeds 1.0. R_0 indicates the average number of secondary cases produced by an infected individual in a completely susceptible population and scenarios where $R_0 > 1.0$ can lead to an epidemic. The non-convex optimization shows that in this scenario it is worth incurring the vaccination cost to mitigate the costs associated with the burden of disease in an epidemic.

7.3.3 Optimal Execution

In this domain we examine the sensitivity of an optimal portfolio transaction model to its parameters. Institutional investors often want to transact a number of shares that exceeds available liquidity. In these situations they face a clear trade-off between the market impact of transacting immediately and the volatility of slow execution. The domain is specified as follows $\mathcal{S} = \langle p, inv \rangle$, where p is the price of the asset and inv is the inventory remaining. $\mathcal{A} \in \{\pi(\theta)\}$, where $\theta \in (0.0, 1.0)$ is the proportion



(a) $V^{\pi^*}(\beta, \nu; s, i, r, \lambda, cost_{inf}, cost_{vaccine})$.



(b) Optimal value of ν for $\beta \in [0.0, 1.0]$.

Figure 7.2: Value function and Nonlinear optimization for the Influenza Public Health Policy domain.

of inventory to be sold. The transition function $\mathcal{T}$ for each state variable in $\mathcal{S}$ is given by:

$$\begin{aligned}\mathcal{T}(p'|p, inv, \pi(\theta)) &= \delta [p' - (p - \kappa \cdot (inv \cdot \pi(\theta)) + \epsilon)] \\ \mathcal{T}(inv'|p, inv, \pi(\theta)) &= \delta [inv' - (inv - inv \cdot \pi(\theta))]\end{aligned}$$

where $\kappa > 0$ is a market-impact parameter and ϵ is a discrete noise parameter. The reward is specified by $\mathcal{R}(p', inv, \pi(\theta)) = p' \cdot inv \cdot \pi(\theta)$.

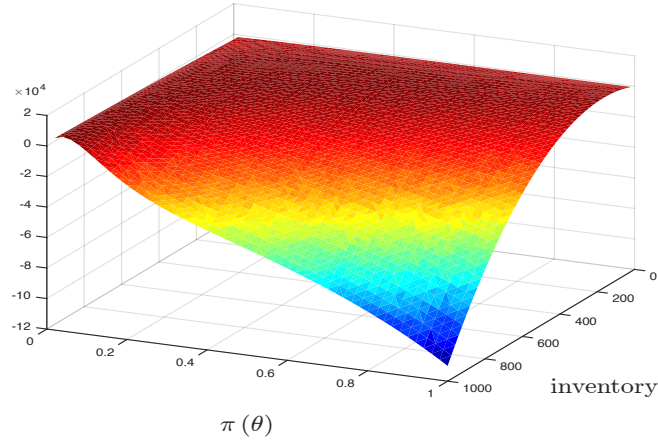
Figures (7.3a) and (7.3b) show the optimal value function at $\mathcal{H} = 10$ and its derivative with respect to the parameter θ , respectively. It is clearly evident that the optimal proportion of shares to be sold is inversely proportional to the amount of inventory remaining. When inventory is low, selling a large proportion of shares allows the investor to capture the current price without having too adverse an effect on future prices. This phenomenon is reversed when inventory is high, with the investor selling a lower proportion of shares to capture a more stable set of future prices by minimising the market impact of their trades. This insight is confirmed in Figure (7.3b) which shows that the value function is most sensitive to θ when the inventory is high.

7.4 Related Work

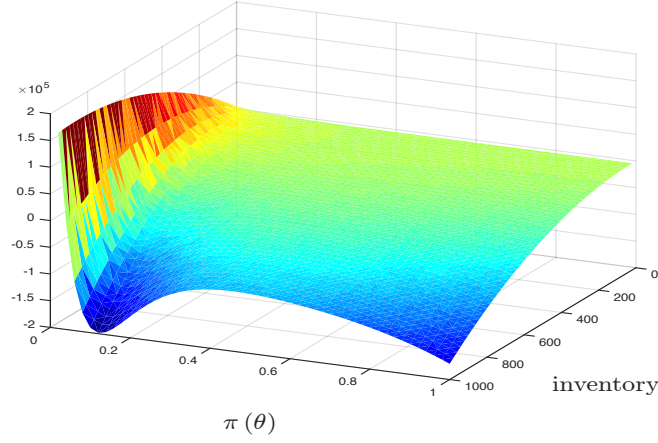
In this chapter, we present Parameterised Hybrid MDPs as a unifying framework, which allows for multi-objective reasoning, exact sensitivity analysis and nonlinear parameterised policy optimization. While there are no comparable parameterised hybrid frameworks that admit exact nonlinear solutions as contributed in this thesis, we briefly survey prior art in each of these three areas and conclude with a discussion of alternate uses of the term *parameterised* in the MDP literature that contrasts with our contributions in this work.

The Multi-objective MDP (MOMDP) literature studies the solution and analysis of trade-offs in the presence of multiple reward criteria. The techniques used to solve MOMDPs with unknown preferences depend on the nature of the scalarisation function used to weight each reward component [Roijers et al., 2013]. If the scalarisation function is linear, then methods such as the Convex Hull Value Iteration algorithm [Barrett and Narayanan, 2008], which returns the optimal policy for discrete *enumerated state* MOMDPs with any linear preference function, can be used. For nonlinear scalarisation functions the Pareto front of a set of undominated policies must be returned. The Pareto front can be prohibitively large and as a result solution techniques such as those of [Chatterjee et al., 2006] and [Pirodda et al., 2015] have focused on approximating the Pareto front, or using a refinement of Pareto dominance known as Lorenz optimality [Perny et al., 2013], to further restrict the size of the solution set. In this chapter we present *exact factored hybrid* MOMDP solutions via the framework of PHMDPs and SDP.

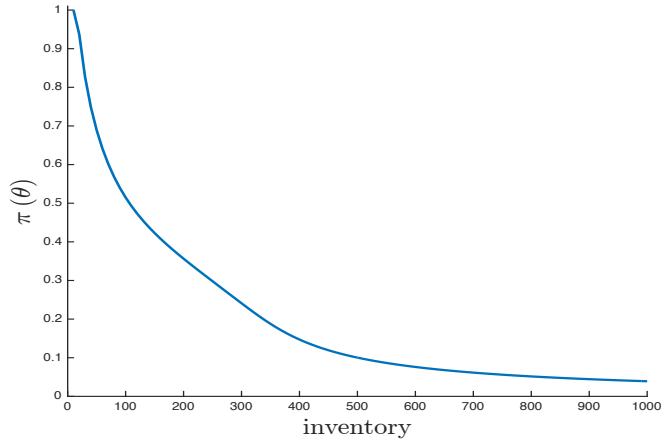
Sensitivity analysis of MDP parameters is a critical tool in understanding what



(a) $V^{\pi^*}(\theta, inv; p = 55.0, \kappa = 0.165)$.



(b) $\nabla_{\theta} V^{\pi^*}(\theta, inv; p = 55.0, \kappa = 0.165)$.



(c) Optimal value of θ for inventory $\in (0.0, 1000.0)$.

Figure 7.3: Value function and Nonlinear optimization for the Optimal Execution domain.

parameters are most important to measure well and how MDP policies vary over a range of parameters. To date, most of the work in this area has focused on uncertainty within the specification of the transition function [Kalyanasundaram et al., 2004], reward function [Tan and Hartman, 2011, Hopp, 1988], or a combination of both [Givan et al., 2000], in discrete MDPs. The framework that we introduce in this thesis enables *exact* sensitivity analysis for PHMDPs that allows it to be applied in continuous state settings and permits the derivation and analysis of the *optimal* policy as a function of these parameters.

Policy gradient methods rely upon optimizing parameterised policies with respect to the expected return by gradient descent. The main concern of such methods is obtaining a good estimator of the policy gradient [Peters and Schaal, 2006]. Two of the most prominent approaches have been the finite-difference methods, such as those of [Ng and Jordan, 2000] and Monte Carlo methods such as [Sutton et al., 1999, Baxter and Bartlett, 2000], both of which are numerically oriented and sample based. Our use of PHMDPs and SDP allows for us to solve for an *exact* policy value as a parameterised function of policy parameters allowing the direct application of (non-)convex optimization to solve for globally optimal policy parameters. This avoids the local optimality and stochasticity associated with traditional policy gradient methods.

Finally, as a point of differentiation from other uses of the term *parameterised* in the MDP literature, we remark that other works [Doshi-Velez and Konidaris, 2016, Duff, 2002, Dearden et al., 1999, Gopalan and Mannor, 2015] have used Parameterised MDP to refer to MDPs with latent parameters whose beliefs can be updated by observing reward and transition samples. In contrast, in this work we assume strict uncertainty of continuous MDP parameters in models that are otherwise fully specified; in this way we can treat parameters simply as free variables that can be parametrically analysed via recent advances in symbolic solution methods and non-convex optimizers [Gao et al., 2013].

7.5 Summary

In this chapter we introduced Parameterised Hybrid MDPs as a unifying framework, which enables the inverse learning of parameters of multi-objective rewards, the examination of parameter sensitivity and the non-convex optimization of continuous policy parameters. We also presented a novel algorithm to solve PHMDPs by utilizing a parametric extension of symbolic dynamic programming and state-of-the-art non-convex optimizers. We demonstrated the utility and scalability of our framework by calculating the first known exact solutions to the inverse learning of parameters for multi-objective navigation, non-convex optimization of vaccination policies and sensitivity analysis of trading models

Conclusions

Decision making is an inescapable requirement of every day life. Formal models of decision making within economics, psychology and decision theoretic planning, a sub-field of artificial intelligence, can be used to determine an optimal sequence of decisions in complex and uncertain environments. Financial markets, replete with their size, complexity and volatility, present a natural proving ground for many sequential decision making algorithms. In this thesis we have provided exact and closed-form solutions to *continuous* state formulations of Markov Decision Processes and Partially Observable Markov Decision Processes, two expressive frameworks for expressing uncertainty in complex environments.

8.1 Summary of Contributions

We briefly summarize our contributions as the following:

1. We are the first to address the open question of sequentially optimal market-making with inventory based reasoning. We introduce a novel continuous state POMDP framework, which is the first to solve, exactly and in closed-form, the optimal market-making problem with inventory, fixed asset value, arbitrary belief state priors, trader models and reward functions. We also provide a novel SDP extension of Point-based Value Iteration algorithms to efficiently derive a value function and policy for this class of continuous state POMDPs. Furthermore, we demonstrate that our sequential market-making model is sequentially optimal, utilises inventory based reasoning, works with flexible trader models and is computationally tractable.
2. We characterise a subclass of zero-sum continuous state stochastic games with restricted reward and transition functions that can be solved exactly via parameterised linear optimisation. We provide an algorithm that solves this subclass of stochastic games exactly and optimally using SDP. Furthermore, we demonstrate the efficacy of our solution in an idealised binary option trading game between a trader and an adversarial market.
3. We have proposed a novel SDP solution to a class of Parameterised Hybrid MDPs, which enables the inverse learning of parameters of multi-objective re-

wards, parameter sensitivity analysis and non-convex analysis and optimization of policy parameters. Our algorithm solves this class of PHMDPs exactly and in closed-form by defining a parameterised variant of SDP. We also demonstrate how our construction of PHMDPs and solution algorithm can be used to model and solve an optimal execution challenge in financial markets.

In the next section we detail some open avenues of research.

8.2 Future Work

This thesis has covered exact symbolic solutions to large subclasses of MDPs and POMDPs in both continuous and discrete state $\mathcal{S}$ formulations. Despite this, there are a number of avenues for further exploration. The MDP and POMDP frameworks utilised in this thesis can be extended to new-subclasses by incorporating a combination of multi-agent settings, continuous action and, in the case of POMDPs, continuous observations. These framework extensions require the development of new symbolic dynamic programming based solution techniques. The final avenue for further research lies in improving the tractability of operations on XADDs, the data structure underlying symbolic dynamic programming.

8.2.1 Sequential Decision Making Frameworks

In this thesis we have introduced new subclasses of the MDP and POMDP frameworks, which allow for more realistic formulations of sequential decision making in financial settings. As mentioned in the previous section, the novel subclasses use a combination of continuous state $\mathcal{S}$, specific reward structures and free parameters. A conspicuous extension to this work is to explore more expressive MDP and POMDP frameworks by using a combination of continuous state $\mathcal{S}$, action $\mathcal{A}$ and, in the case of POMDPs, continuous observation $\mathcal{O}$. Multi-agent extensions to the aforementioned MDP and POMDP subclasses can also be researched. We will now discuss each of these novel directions in turn.

The MDP and POMDP models covered in this thesis can be extended to hybrid continuous/discrete action $\mathcal{A}$. The MDP formulation of zero-sum continuous stochastic games can be extended to incorporate continuous actions by leveraging symbolic constrained optimisation to solve the continuous action maximisation problem [Zamani et al., 2012a]. The resulting subclass of MDPs would permit more expressive formulations of a wider range of financial domains, one such example being a binary option game where the actions are the price at which to trade with the market. Continuous action extensions to the POMDP market maker are less straight forward. We surmise that it is possible to calculate an optimal partitioning of the continuous action space into a discrete number of slices.

The majority of the MDP and POMDP models presented in this thesis have been from a single agent perspective, where an agent executes decisions without regard to the presence of other actors in the environment. The obvious exception to this is

the zero-sum continuous stochastic game MDP, where an adversarial agent is represented as the environment itself. A fruitful avenue of research lies in extending the multi-agent frameworks presented Section 2.3.2 into continuous state $\mathcal{S}$ formulations, which can then be solved using the SDP algorithms presented in this thesis. Two such frameworks are the Decentralised Markov Decision Process and Decentralised Partially Observable Markov Decision Process, in which the aim is to calculate a joint policy for all agents under a common reward function. The DEC-MDP and DEC-POMDP frameworks lend themselves to the representation of a financial trading system operating across multiple markets. Each agent in the DEC-MDP or DEC-POMDP trading system executes orders in its own market, independently of other agents. The profitability of the trading system as a whole is then measured by the sum of the performance of each agent in their respective markets.

8.2.2 Solution Algorithms

This thesis has presented Value Iteration and its variants as the primary dynamic programming approach to solve continuous state $\mathcal{S}$ formulations of MDPs and POMDPs. An alternate avenue of research is to investigate symbolic Policy Iteration, which may have quicker convergence and better computational tractability. Raghavan [Raghavan et al., 2013] proposed a symbolic implementation of Modified Policy Iteration, known as Opportunistic Policy Iteration (OPI), which lies between traditional value iteration and Modified Policy Iteration. OPI performs Value Iteration backups and also applies policy constraints if it does not increase the size of the value function representation. Extending OPI to factored observation spaces $\mathcal{O}$ in the POMDP setting may prove fruitful in solving more complex domains.

The novel solution algorithms introduced in this thesis have focussed on calculating policies for the entire continuous state space $\mathcal{S}$. An alternate approach is to consider *local* solution methods which, as introduced in Section 3.3, calculate policies for a subset of the state space. As mentioned in Section 3.3, the dominant paradigm for local solutions is that of Monte Carlo Tree Search. MCTS has made significant contributions to solving large MDPs [Kocsis and Szepesvári, 2006] and POMDPs [Silver and Veness, 2010]. However these algorithms dealt with only discrete state $\mathcal{S}$, discrete action $\mathcal{A}$ and, in the case of POMCP, discrete observation $\mathcal{O}$ formulations. Extending these algorithms to continuous state $\mathcal{S}$ and/or continuous action $\mathcal{A}$ settings is an open area of research. Hierarchical Optimistic Optimization applied to Trees (HOOT) [Mansley et al., 2011] leverages Hierarchical Optimistic Optimisation (HOO) [Bubeck et al., 2011] to extend UCT to continuous action $\mathcal{A}$ spaces. However, the *continuous* state $\mathcal{S}$ setting remains unexplored. SDP may be leveraged in such a setting to represent the continuous state space $\mathcal{S}$. POMCP can also be extended into continuous action $\mathcal{A}$ and continuous state $\mathcal{S}$ settings by leveraging HOO as the underlying bandit algorithm and SDP to represent the state space. While MCTS no longer calculates *exact* global policies when compared to the algorithms presented in Chapters 5 to 7, they do alleviate the computational burden of calculating global solutions.

8.2.3 SDP Data Structures

In this thesis we have utilised Extended Algebraic Decision Diagrams as the data structure underlying Symbolic Dynamic Programming. In theory, XADD nodes are able to represent any function. However, operations on XADDs as part of SDP leads to an exponential growth in the number of nodes required to represent non-linear functions. Two avenues for further investigation are to: (i) prune and reduce the size of the resulting XADDs; or (ii) compress them in exchange for a bounded approximation error [Vianna et al., 2013]. Pruning of XADD nodes and branches is possible in the case of linear constraints, but is computationally intractable beyond this simple case. Investigating methods of handling polynomial function representations, that may occur in XADD decision nodes, via δ -satisfiability solvers such as dReal [Gao et al., 2013], or other methods which provide upper or lower bounded approximations, are an interesting research topic which can potentially alleviate the current limits on SDP within large MDPs and POMDPs.

In summary, this thesis has contributed novel Symbolic Dynamic Programming solution techniques to a variety of MDP and POMDP frameworks. The solution techniques allowed the calculation of optimal policies for a zero-sum binary option trading scenario and sequentially optimal market-making as well as the examination of the sensitivity of an optimal transaction execution model to its parameters. We hope that the advances made in this thesis open a number of potential novel research directions that we believe can help progress optimal sequential decision making within financial markets.

References

- [Abernethy and Kale, 2013] Abernethy, J. and Kale, S. (2013). Adaptive Market Making via Online Learning. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems* 26, NIPS, pages 2058–2066. Curran Associates, Inc.
- [Agrawal, 1995] Agrawal, R. (1995). The Continuum-Armed Bandit Problem. *SIAM Journal on Control and Optimization*, 33(6):1926–1951.
- [Allais, 1953] Allais, M. (1953). Le comportement de l’homme rationnel devant le risque: Critique des postulats et axiomes de l’ecole americaine. *Econometrica*, 21(4):503–546.
- [Amihud and Mendelson, 1980] Amihud, Y. and Mendelson, H. (1980). Dealership Market: Market-Making with Inventory. *Journal of Financial Economics*, 8(1):31–53.
- [Auer et al., 2002] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2):235–256.
- [Auer et al., 2003] Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (2003). The Nonstochastic Multiarmed Bandit Problem. *SIAM Journal on Computing*, 32(1):48–77.
- [Barrett and Narayanan, 2008] Barrett, L. and Narayanan, S. (2008). Learning all optimal policies with multiple criteria. In McCallum, A. and Roweis, S., editors, *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, ICML, pages 41–47. Omnipress.
- [Barro and Ursúa, 2017] Barro, R. J. and Ursúa, J. F. (2017). Stock-market crashes and depressions. *Research in Economics*, 71(3):384 – 398.
- [Barto et al., 1995] Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to Act using Real-Time Dynamic Programming. *Artificial Intelligence*, 72(1-2):81–138.
- [Baxter and Bartlett, 2000] Baxter, J. and Bartlett, P. L. (2000). Direct gradient-based reinforcement learning. In *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353)*, volume 3 of CS, pages 271–274. IEEE.
- [Beckert and Berghoff, 2013] Beckert, J. and Berghoff, H. (2013). Risk and uncertainty in financial markets: a symposium. *Socio-Economic Review*, 11(3):497–499.

-
- [Bellman, 1957] Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.
- [Bennett and Mangasarian, 1993] Bennett, K. P. and Mangasarian, O. L. (1993). Bilinear Separation of Two Sets in N-space. *Computational Optimization and Applications*, 2(3):207–227.
- [Bernstein et al., 2000] Bernstein, D. S., Zilberstein, S., and Immerman, N. (2000). The complexity of decentralized control of markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI, pages 32–37, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Bertsekas, 1987] Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- [Bertsekas, 2000] Bertsekas, D. P. (2000). *Dynamic Programming and Optimal Control*. Athena Scientific, 2 edition.
- [Bertsekas and Tsitsiklis, 1989] Bertsekas, D. P. and Tsitsiklis, J. N. (1989). *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- [Boutilier, 1999] Boutilier, C. (1999). Sequential optimality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, IJCAI '99, pages 478–485, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Boutilier et al., 1999] Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1–94.
- [Boutilier et al., 1995] Boutilier, C., Dearden, R., and Goldszmidt, M. (1995). Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 2 of *IJCAI*, pages 1104–1111, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Boutilier and Poole, 1996] Boutilier, C. and Poole, D. (1996). Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume 2 of *AAAI*, pages 1168–1175. AAAI Press.
- [Boutilier et al., 2001] Boutilier, C., Reiter, R., and Price, B. (2001). Symbolic Dynamic Programming for First-Order MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, IJCAI, pages 690–697. Morgan Kaufmann Publishers Inc.
- [Brahma et al., 2012] Brahma, A., Chakraborty, M., Das, S., Lavoie, A., and Magdon-Ismail, M. (2012). A Bayesian Market Maker. In *Proceedings of the Thirteenth ACM Conference on Electronic Commerce*, EC, pages 215–232. ACM Press.

-
- [Braziunas, 2003] Braziunas, D. (2003). POMDP solution methods: a survey. Technical report, Department of Computer Science, University of Toronto.
- [Bresina et al., 2002] Bresina, J., Dearden, R., Meuleau, N., Ramakrishnan, S., Smith, D., and Washington, R. (2002). Planning Under Continuous Time and Resource Uncertainty: A Challenge for AI. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, UAI'02, pages 77–84, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Bubeck et al., 2011] Bubeck, S., Munos, R., Stoltz, G., and Szepesvári, C. (2011). $\mathcal{X}$ -armed bandits. *Journal of Machine Learning Research*, 12:1655–1695.
- [Cassandra et al., 1997] Cassandra, A. R., Littman, M. L., and Zhang, N. L. (1997). Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes. In *Proceedings of the Thirteenth Conference Annual Conference on Uncertainty in Artificial Intelligence*, UAI, pages 54–61. Morgan Kaufmann.
- [Chakraborty and Kearns, 2011] Chakraborty, T. and Kearns, M. (2011). Market Making and Mean Reversion. In *Proceedings of the 12th ACM conference on Electronic Commerce*, EC, pages 307–314.
- [Chatterjee et al., 2006] Chatterjee, K., Majumdar, R., and Henzinger, T. A. (2006). *Markov Decision Processes with Multiple Objectives*, pages 325–336. STACS. Springer.
- [Chen and Pennock, 2007] Chen, Y. and Pennock, D. (2007). A Utility Framework for Bounded-loss Market Makers. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, UAI, pages 49–56.
- [Coulom, 2007] Coulom, R. (2007). *Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search*, pages 72–83. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [Das, 2005] Das, S. (2005). A Learning Market-Maker in the Glosten-Milgrom Model. *Quantitative Finance*, 5(2):169–180.
- [Das, 2008] Das, S. (2008). The Effects of Market-Making on Price Dynamics. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pages 887–894. International Foundation for Autonomous Agents and Multiagent Systems.
- [Das and Magdon-Ismail, 2009] Das, S. and Magdon-Ismail, M. (2009). Adapting to a Market Shock: Optimal Sequential Market-Making. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 22*, NIPS, pages 361–368. Curran Associates, Inc.
- [De Bondt and Thaler, 1985] De Bondt, W. F. M. and Thaler, R. (1985). Does the Stock Market Overreact? *The Journal of Finance*, 40(3):793–805.
- [Dean and Kanazawa, 1989] Dean, T. and Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150.

-
- [Dearden et al., 1999] Dearden, R., Friedman, N., and Andre, D. (1999). Model Based Bayesian Exploration. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, UAI*, pages 150–159.
- [d’Epenoux, 1963] d’Epenoux, F. (1963). A Probabilistic Production and Inventory Problem. *Management Science*, 10(1):98–108.
- [Derman, 1970] Derman, C. (1970). *Finite State Markovian Decision Processes*. Academic Press, Inc., Orlando, FL, USA.
- [Doshi-Velez and Konidaris, 2016] Doshi-Velez, F. and Konidaris, G. (2016). Hidden Parameter Markov Decision Processes: A Semiparametric Regression Approach for Discovering Latent Task Parametrizations. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1432–1440.
- [Duff, 2002] Duff, M. O. (2002). *Optimal Learning: Computational procedures for Bayes-adaptive Markov Decision Processes*. PhD thesis, University of Massachusetts Amherst.
- [Feng et al., 2004] Feng, Z., Dearden, R., Meuleau, N., and Washington, R. (2004). Dynamic programming for structured continuous markov decision problems. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence, UAI*, pages 154–161, Arlington, Virginia, United States. AUAI Press.
- [Frydman and Camerer, 2016] Frydman, C. and Camerer, C. F. (2016). The Psychology and Neuroscience of Financial Decision Making. *Trends in Cognitive Sciences*, 20(9):661 – 675.
- [Gao et al., 2013] Gao, S., Kong, S., and Clarke, E. M. (2013). *dReal: An SMT Solver for Nonlinear Theories over the Reals*, pages 208–214. Springer Berlin Heidelberg.
- [Gittins, 1979] Gittins, J. C. (1979). Bandit Processes and Dynamic Allocation Indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(2):148–177.
- [Givan et al., 2000] Givan, R., Leach, S., and Dean, T. (2000). Bounded-parameter Markov Decision Processes. *Artificial Intelligence*, 122(1–2):71–109.
- [Glosten and Milgrom, 1985] Glosten, L. R. and Milgrom, P. R. (1985). Bid, Ask and Transaction Prices in a Specialist Market with Heterogeneously Informed Traders. *Journal of Financial Economics*, 14(1):71–100.
- [Gopalan and Mannor, 2015] Gopalan, A. and Mannor, S. (2015). Thompson Sampling for Learning Parameterized Markov Decision Processes. In *Proceedings of The 28th Conference on Learning Theory, COLT*, pages 861–898.
- [Guestrin et al., 2003] Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. (2003). Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19(1):399–468.

-
- [Guo and Liu, 2001] Guo, X. and Liu, K. (2001). A note on optimality conditions for continuous-time Markov Decision Processes with average cost criterion. *IEEE Transactions on Automatic Control*, 46(12):1984–1989.
- [Hansen, 1998] Hansen, E. A. (1998). Solving POMDPs by Searching in Policy Space. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI’98, pages 211–219, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Hansen and Zilberstein, 2001] Hansen, E. A. and Zilberstein, S. (2001). LAO*: A Heuristic Search Algorithm that Finds Solutions with Loops. *Artificial Intelligence*, 129(1):35–62.
- [Hanson, 2003] Hanson, R. (2003). Combinatorial Information Market Design. *Information Systems Frontiers*, 5(1):107–119.
- [Hanson, 2007] Hanson, R. (2007). Logarithmic Market Scoring Rules for Modular Combinatorial Information Aggregation. *Journal of Prediction Markets*, 1(1):3–15.
- [Heffernan et al., 2005] Heffernan, J., Smith, R., and Wahl, L. (2005). Perspectives on the Basic Reproductive Ratio. *Journal of The Royal Society Interface*, 2(4):281–293.
- [Hoey and Poupart, 2005] Hoey, J. and Poupart, P. (2005). Solving pomdps with continuous or large discrete observation spaces. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, IJCAI’05, pages 1332–1338, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Hoey et al., 1999] Hoey, J., St-Aubin, R., Hu, A., and Boutilier, C. (1999). Spudd: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI, pages 279–288, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Hopp, 1988] Hopp, W. J. (1988). Sensitivity Analysis in Discrete Dynamic Programming. *Journal of Optimization Theory and Applications*, 56(2):257–269.
- [Howard, 1960] Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. The MIT press.
- [Hu and Wellman, 1998] Hu, J. and Wellman, M. P. (1998). Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML, pages 242–250, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Johnson et al., 2013] Johnson, N., Zhao, G., Hunsader, E., Qi, H., Johnson, N., Meng, J., and Tivnan, B. (2013). Abrupt Rise of New Machine Ecology Beyond Human Response Time. *Scientific Reports*, 3:2627.

-
- [Kaelbling et al., 1998] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101:99–134.
- [Kahneman and Frederick, 2002] Kahneman, D. and Frederick, S. (2002). Representativeness Revisited: Attribute Substitution in Intuitive Judgment. In Gilovich, T., Griffin, D., and Kahneman, D., editors, *Heuristics & Biases: The Psychology of Intuitive Judgment.*, pages 49–81. New York. Cambridge University Press.
- [Kalyanasundaram et al., 2004] Kalyanasundaram, S., Chong, E. K. P., and Shroff, N. B. (2004). Markov decision processes with uncertain transition rates: sensitivity and max hyphen min control. *Asian Journal of Control*, 6(2):253–269.
- [Kaspi and Mandelbaum, 1998] Kaspi, H. and Mandelbaum, A. (1998). Multi-armed bandits in Discrete and Continuous time. *The Annals of Applied Probability*, 8(4):1270–1290.
- [Kearns et al., 2002] Kearns, M., Mansour, Y., and Ng, A. Y. (2002). A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. *Machine Learning*, 49(2):193–208.
- [Kearns and Koller, 1999] Kearns, M. J. and Koller, D. (1999). Efficient Reinforcement Learning in Factored MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 740–747.
- [Keller and Helmert, 2013] Keller, T. and Helmert, M. (2013). Trial-based Heuristic Tree Search for Finite Horizon MDPs. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS*, pages 135–143. AAAI Press.
- [Kermack and McKendrick, 1927] Kermack, W. O. and McKendrick, A. G. (1927). A Contribution to the Mathematical Theory of Epidemics. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 115(772):700–721.
- [Kleinberg, 2004] Kleinberg, R. (2004). Nearly Tight Bounds for the Continuum-armed Bandit Problem. In *Advances in Neural Information Processing Systems 17, NIPS*, pages 697–704, Cambridge, MA, USA. MIT Press.
- [Kocsis and Szepesvári, 2006] Kocsis, L. and Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. In Fürnkranz, J., Scheffer, T., and Spiliopoulou, M., editors, *Proceedings of the 17th European Conference on Machine Learning, ECML*, pages 282–293, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Koller and Parr, 1999] Koller, D. and Parr, R. (1999). Computing factored value functions for policies in structured mdps. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, volume 2 of *IJCAI*, pages 1332–1339, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

-
- [Lai and Robbins, 1985] Lai, T. and Robbins, H. (1985). Asymptotically Efficient Adaptive Allocation Rules. *Advances in Applied Mathematics*, 6(1):4–22.
- [Li and Littman, 2005] Li, L. and Littman, M. L. (2005). Lazy Approximation for Solving Continuous Finite-horizon MDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, volume 3 of AAAI, pages 1175–1180. AAAI Press.
- [Littman, 1994] Littman, M. L. (1994). Markov Games as a Framework for Multi-Agent Reinforcement Learning. In *Proceedings of the Eleventh International Conference on Machine Learning Machine Learning*, ICML, pages 157–163. Morgan Kaufmann Publishers Inc.
- [Mandelbrot, 1963] Mandelbrot, B. (1963). The Variation of Certain Speculative Prices. *The Journal of Business*, 36.
- [Mansley et al., 2011] Mansley, C. R., Weinstein, A., and Littman, M. L. (2011). Sample-based Planning for Continuous Action Markov Decision Processes. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling*, ICAPS, pages 335–338, Freiburg, Germany. AAAI Press.
- [Meuleau et al., 2009] Meuleau, N., Benazera, E., Brafman, R. I., Hansen, E. A., and Mausam (2009). A Heuristic Search Approach to Planning with Continuous Resources in Stochastic Domains. *Journal of Artificial Intelligence Research*, 34:27–59.
- [Mohan, 2010] Mohan, R. (2010). A Contextual-bandit Approach to Personalized News Article Recommendation. In *Proceedings of WWW 2010: Proceedings of the 19th international conference on World Wide Web*, Raleigh, NC, USA.
- [Monahan, 1982] Monahan, G. E. (1982). A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *Management Science*, 28(1):1–16.
- [Neumann, 1928] Neumann, J. (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320.
- [Neumann and Morgenstern, 1944] Neumann, J. V. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press.
- [Ng and Jordan, 2000] Ng, A. Y. and Jordan, M. (2000). PEGASUS: A Policy Search Method for Large MPDs and POMDPs. In *Proceedings of the Sixteenth conference on Uncertainty in Artificial Intelligence*, UAI, pages 406–415, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Ng and Russell, 2000] Ng, A. Y. and Russell, S. J. (2000). Algorithms for Inverse Reinforcement Learning. In *ICML*, ICML, pages 663–670.
- [Osborne, 2004] Osborne, M. J. (2004). *An Introduction to Game Theory*. Oxford University Press.

-
- [Othman and Sandholm, 2012] Othman, A. and Sandholm, T. (2012). Rational Market Making with Probabilistic Knowledge. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, volume 2 of AAMAS, pages 645–652. International Foundation for Autonomous Agents and Multiagent Systems.
- [Othman et al., 2010] Othman, A., Sandholm, T., Pennock, D., and Reeves, D. (2010). A Practical Liquidity-sensitive Automated Market Maker. In *Proceedings of the 11th ACM conference on Electronic Commerce, EC*, pages 377–386. ACM Press.
- [Papadimitriou and Tsitsiklis, 1987] Papadimitriou, C. and Tsitsiklis, J. N. (1987). The Complexity of Markov Decision Processes. *Mathematics of Operations Research*, 12(3):441–450.
- [Pennock and Sami, 2007] Pennock, D. and Sami, R. (2007). *Algorithmic Game Theory*, chapter Computational Aspects of Prediction Markets. AGT. Cambridge University Press.
- [Perny et al., 2013] Perny, P., Weng, P., Goldsmith, J., and Hanna, J. P. (2013). Approximation of Lorenz-Optimal Solutions in Multiobjective Markov Decision Processes. In *Workshops at the Twenty Seventh AAAI Conference on Artificial Intelligence*, AAAI.
- [Perold, 1988] Perold, A. F. (1988). The Implementation Shortfall: Paper versus Reality. *Journal of Portfolio Management*, 14(3):4–9.
- [Peters and Schaal, 2006] Peters, J. and Schaal, S. (2006). Policy Gradient Methods for Robotics. In *Intelligent Robots and Systems, IRS*, pages 2219–2225. IEEE.
- [Petrik and Zilberstein, 2011] Petrik, M. and Zilberstein, S. (2011). Robust Approximate Bilinear Programming for Value Function Approximation. *The Journal of Machine Learning Research*, 12:3027–3063.
- [Pineau et al., 2003] Pineau, J., Gordon, G. J., and Thrun, S. (2003). Point-based Value Iteration: An Anytime Algorithm for POMDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1025–1030. Morgan Kaufmann Publishers Inc.
- [Pirotta et al., 2015] Pirotta, M., Parisi, S., and Restelli, M. (2015). Multi-Objective Reinforcement Learning with Continuous Pareto Frontier Approximation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI*, pages 2928–2934.
- [Porta et al., 2005] Porta, J., T. J. Spaan, M., and Vlassis, N. (2005). Robot Planning in Partially Observable Continuous Domains. In *IEEE Transactions on Pattern Analysis and Machine Intelligence - PAMI*, pages 217–224.

-
- [Porta et al., 2006] Porta, J. M., Vlassis, N., Spaan, M. T., and Poupart, P. (2006). Point-Based Value Iteration for Continuous POMDPs. *Journal of Machine Learning Research*, 7:2329–2367.
- [Poupart, 2005] Poupart, P. (2005). *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. PhD thesis, Department of Computer Science, University of Toronto, Toronto.
- [Poupart et al., 2006] Poupart, P., Vlassis, N., Hoey, J., and Regan, K. (2006). An Analytic Solution to Discrete Bayesian Reinforcement Learning. In *Proceedings of the 23rd International Conference on Machine learning, ICML*, pages 697–704, New York, NY, USA. ACM.
- [Puterman, 1994] Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.
- [Raghavan et al., 2013] Raghavan, A., Khardon, R., Fern, A., and Tadepalli, P. (2013). Symbolic opportunistic policy iteration for factored-action mdps. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 2499–2507. Curran Associates, Inc.
- [Robbins, 1952] Robbins, H. (1952). Some Aspects of the Sequential Design of Experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535.
- [Roijsers et al., 2013] Roijsers, D. M., Vamplew, P., Whiteson, S., and Richard Dazeley (2013). A Survey of Multi-Objective Sequential Decision-Making. *Journal of Artificial Intelligence Research*, 48:67–113.
- [Ross et al., 2008] Ross, S., Pineau, J., Paquet, S., and Chaib-draa, B. (2008). On-line Planning Algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32(1):663–704.
- [Sanner and Abbasnejad, 2012] Sanner, S. and Abbasnejad, E. (2012). Symbolic variable elimination for discrete and continuous graphical models. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12)*, Toronto, Canada.
- [Sanner et al., 2011] Sanner, S., Delgado, K., and Nunes de Barros, L. (2011). Symbolic Dynamic Programming for Discrete and Continuous State MDPs. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, UAI*, pages 643–652. AUAI Press.
- [Schwartz, 1991] Schwartz, R. A. (1991). *Reshaping the Equity Markets: A Guide for the 1990s*. Business One Irwin, New York, NY.
- [Shapley, 1953] Shapley, L. S. (1953). Stochastic Games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100.

-
- [Silver and Veness, 2010] Silver, D. and Veness, J. (2010). Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems 23*, pages 2164–2172.
- [Simon, 1991] Simon, H. A. (1991). Bounded Rationality and Organizational Learning. *Organization Science*, 2(1):125–134.
- [Sondik, 1971] Sondik, E. J. (1971). *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford University.
- [Sondik, 1978] Sondik, E. J. (1978). The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304.
- [Spaan and Vlassis, 2005] Spaan, M. T. J. and Vlassis, N. (2005). Perseus: Randomized Point-Based Value Iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220.
- [St-Aubin et al., 2000] St-Aubin, R., Hoey, J., and Boutilier, C. (2000). APRICODD: Approximate Policy Construction Using Decision Diagrams. In *Advances in Neural Information Processing Systems 13*, NIPS, pages 1089–1095, Denver, Colorado, USA. MIT Press.
- [Sutton et al., 1999] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems 12*, NIPS, pages 1057–1063. MIT Press.
- [Sweller, 1988] Sweller, J. (1988). Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science*, 12(2):257–285.
- [Szepesvári and Littman, 1996] Szepesvári, C. and Littman, M. L. (1996). Generalized markov decision processes: Dynamic-programming and reinforcement-learning algorithms. In Saitta, L., editor, *Machine Learning: Proceedings of the Thirteenth International Conference*. Morgan Kaufmann.
- [Tan and Hartman, 2011] Tan, C. H. and Hartman, J. C. (2011). Sensitivity Analysis in Markov Decision Processes with Uncertain Reward Parameters. *Journal of Applied Probability*, 48(4):954–967.
- [The MathWorks Inc., 2010] The MathWorks Inc. (2010). *MATLAB version 8.5.197613 (R2015a) and Optimization Toolbox 7.2*. The MathWorks Inc.
- [Thompson, 1933] Thompson, W. R. (1933). On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25(3/4):285–294.
- [Tversky and Kahneman, 1974] Tversky, A. and Kahneman, D. (1974). Judgment under Uncertainty: Heuristics and Biases. *Science*, 185(4157):1124–1131.

-
- [Vianna et al., 2013] Vianna, L. G. R., Sanner, S., and Nunes de Barros, L. (2013). Bounded Approximate Symbolic Dynamic Programming for Hybrid MDPs. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, UAI, pages 1–9, Bellevue, USA.
- [White III and Kim, 1980] White III, C. C. and Kim, K. M. (1980). Solution Procedures for Solving Vector Criterion Markov Decision Processes. *Large Scale Systems*, 1:129–140.
- [Zamani et al., 2012a] Zamani, Z., Sanner, S., and Fang, C. (2012a). Symbolic Dynamic Programming for Continuous State and Action MDPs. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI, pages 1–7, Toronto, Canada. AAAI Press.
- [Zamani et al., 2012b] Zamani, Z., Sanner, S., Poupart, P., and Kersting, K. (2012b). Symbolic Dynamic Programming for Continuous State and Observation POMDPs. In *Advances in Neural Information Processing Systems 25*, NIPS, pages 1403–1411.