Reinforcement Learning in Finance

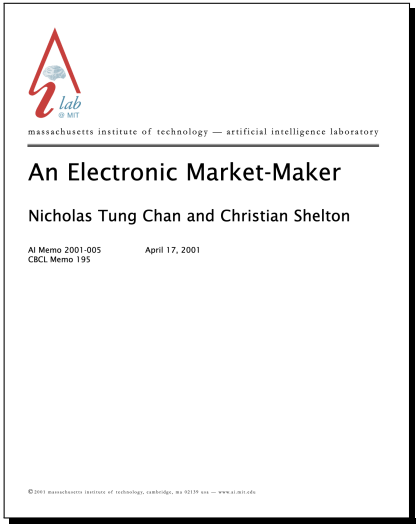
Leveraging Structure (for Fun and Profit)

Thomas Spooner

13th August, 2022

Key Papers: A Timeline

- Chan and Shelton (2001)



massachusetts institute of technology — artificial intelligence laboratory

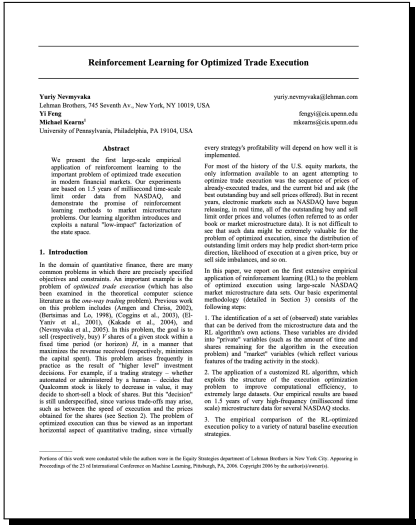
An Electronic Market-Maker

Nicholas Tung Chan and Christian Shelton

AI Memo 2001-005 April 17, 2001
CBCL Memo 195

Key Papers: A Timeline

- Chan and Shelton (2001)
- Nevmyvaka, Feng and Kearns (2006)



Reinforcement Learning for Optimized Trade Execution

Yuriy Nevmyvaka yuriy.nevmyvaka@lehman.com
Lehman Brothers, 745 Seventh Av., New York, NY 10019, USA
Yi Feng fengyi@cis.upenn.edu
Michael Kearns[1] mkearns@cis.upenn.edu
University of Pennsylvania, Philadelphia, PA 19104, USA

Abstract

We present the first large-scale empirical application of reinforcement learning to the important problem of optimized trade execution in modern financial markets. Our experiments are based on 1.5 years of millisecond time-scale limit order data from NASDAQ, and demonstrate the promise of reinforcement learning methods to market microstructure problems. Our learning algorithm introduces and exploits a natural "low-impact" factorization of the state space.

1. Introduction

In the domain of quantitative finance, there are many common problems in which there are precisely specified objectives and constraints. An important example is the problem of *optimized trade execution* (which has also been examined in the theoretical computer science literature as the *one-way trading* problem). Previous work on this problem includes (Amgen and Chriss, 2002), (Bertsimas and Lo, 1998), (Coggins et al., 2003), (El-Yaniv et al., 2001), (Kakade et al., 2004), and (Nevmyvaka et al., 2005). In this problem, the goal is to sell (respectively, buy) V shares of a given stock within a fixed time period (or horizon) H, in a manner that maximizes the revenue received (respectively, minimizes the capital spent). This problem arises frequently in practice as the result of "higher level" investment decisions. For example, if a trading strategy – whether automated or administered by a human – decides that Qualcomm stock is likely to decrease in value, it may decide to short-sell a block of shares. But this "decision" is still underspecified, since various trade-offs may arise, such as between the speed of execution and the prices obtained for the shares (see Section 2). The problem of optimized execution can thus be viewed as an important horizontal aspect of quantitative trading, since virtually every strategy's profitability will depend on how well it is implemented.

For most of the history of the U.S. equity markets, the only information available to an agent attempting to optimize trade execution was the sequence of prices of already-executed trades, and the current bid and ask (the best outstanding buy and sell prices offered). But in recent years, electronic markets such as NASDAQ have begun releasing, in real time, all of the outstanding buy and sell limit order prices and volumes (often referred to as order book or market microstructure data). It is not difficult to see that such data might be extremely valuable for the problem of optimized execution, since the distribution of outstanding limit orders may help predict short-term price direction, likelihood of execution at a given price, buy or sell-side imbalances, and so on.

In this paper, we report on the first extensive empirical application of reinforcement learning (RL) to the problem of optimized execution using large-scale NASDAQ market microstructure data sets. Our basic experimental methodology (detailed in Section 3) consists of the following steps:

1. The identification of a set of (observed) state variables that can be derived from the microstructure data and the RL algorithm's own actions. These variables are divided into "private" variables (such as the amount of time and shares remaining for the algorithm in the execution problem) and "market" variables (which reflect various features of the trading activity in the stock).

2. The application of a customized RL algorithm, which exploits the structure of the execution optimization problem to improve computational efficiency, to extremely large datasets. Our empirical results are based on 1.5 years of very high-frequency (millisecond time scale) microstructure data for several NASDAQ stocks.

3. The empirical comparison of the RL-optimized execution policy to a variety of natural baseline execution strategies.

[1] Portions of this work were conducted while the authors were in the Equity Strategies department of Lehman Brothers in New York City. Appearing in Proceedings of the 23 rd International Conference on Machine Learning, Pittsburgh, PA, 2006.

Key Papers: A Timeline

- Chan and Shelton (2001)
- Nevmyvaka, Feng and Kearns (2006)
- Avellaneda and Stoikov (2008)

Quantitative Finance, Vol. 8, No. 3, April 2008, 217–224

Routledge
Taylor & Francis Group

High-frequency trading in a limit order book

MARCO AVELLANEDA and SASHA STOIKOV*

Mathematics, New York University, 251 Mercer Street, New York, NY 10012, USA

(*Received 24 April 2006; in final form 3 April 2007*)

1. Introduction

The role of a dealer in securities markets is to provide liquidity on the exchange by quoting bid and ask prices at which he is willing to buy and sell a specific quantity of assets. Traditionally, this role has been filled by market-maker or specialist firms. In recent years, with the growth of electronic exchanges such as Nasdaq's Inet, anyone willing to submit limit orders in the system can effectively play the role of a dealer. Indeed, the availability of high frequency data on the limit order book (see www.inetats.com) ensures a fair playing field where various agents can post limit orders at the prices they choose. In this paper, we study the optimal submission strategies of bid and ask orders in such a limit order book.

The pricing strategies of dealers have been studied extensively in the microstructure literature. The two most often addressed sources of risk facing the dealer are (i) the inventory risk arising from uncertainty in the asset's value and (ii) the asymmetric information risk arising from informed traders. Useful surveys of their results can be found in Biais *et al.* (2004), Stoll (2003) and a book by O'Hara (1997). In this paper, we will focus on the inventory effect. In fact, our model is closely related to a paper by Ho and Stoll (1981), which analyses the optimal prices for a monopolistic dealer in a single stock. In their model, the authors specify a 'true' price for the asset, and derive optimal bid and ask quotes around this price, to account for the effect of the inventory. This inventory effect was found to be significant in an empirical study of AMEX Options by Ho and Macris (1984). In another paper by Ho and Stoll (1980), the problem of dealers under competition is analysed and the bid and ask prices are shown to be related to the reservation (or indifference) prices of the agents. In our framework, we will assume that our agent is but one player in the market and the 'true' price is given by the market mid-price.

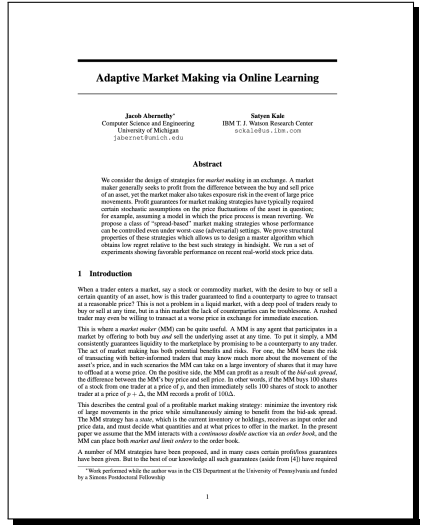
Of crucial importance to us will be the arrival rate of buy and sell orders that will reach our agent. In order

*Corresponding author. Email: sashastoikov@gmail.com

Quantitative Finance
ISSN 1469-7688 print/ISSN 1469-7696 online © 2008 Taylor & Francis
http://www.informaworld.com
DOI: 10.1080/14697680701381228

Key Papers: A Timeline

- Chan and Shelton (2001)
- Nevmyvaka, Feng and Kearns (2006)
- Avellaneda and Stoikov (2008)
- Abernethy and Kale (2013)



Adaptive Market Making via Online Learning

Jacob Abernethy*
Computer Science and Engineering
University of Michigan
jabernet@umich.edu

Satyen Kale
IBM T. J. Watson Research Center
sckale@us.ibm.com

Abstract

We consider the design of strategies for *market making* in an exchange. A market maker generally seeks to profit from the difference between the buy and sell price of an asset, yet the market maker also takes exposure risk in the event of large price movements. Profit guarantees for market making strategies have typically required certain stochastic assumptions on the price fluctuations of the asset in question; for example, assuming a model in which the price process is mean reverting. We propose a class of "spread based" market making strategies whose performance can be controlled even under worst-case (adversarial) settings. We prove structural properties of these strategies which allows us to design a master algorithm which obtains low regret relative to the best such strategy in hindsight. We run a set of experiments showing favorable performance on recent real-world stock price data.

1 Introduction

When a trader enters a market, say a stock or commodity market, with the desire to buy or sell a certain quantity of an asset, how is this trader guaranteed to find a counterparty to agree to transact at a reasonable price? This is not a problem in a liquid market, with a deep pool of traders ready to buy or sell at any time, but in a thin market the lack of counterparties can be troublesome. A rushed trader may even be willing to transact at a worse price in exchange for immediate execution.

This is where a *market maker* (MM) can be quite useful. A MM is any agent that participates in a market by offering to both buy *and* sell the underlying asset at any time. To put it simply, a MM consistently guarantees liquidity to the marketplace by promising to be a counterparty to any trader. The act of market making has both potential benefits and risks. For one, the MM bears the risk of transacting with better-informed traders that may know much more about the movement of the asset's price, and in such scenarios the MM can take on a large inventory of shares that it may have to offload at a worse price. On the positive side, the MM can profit as a result of the bid-ask *spread*, the difference between the MM's buy price and sell price. In other words, if the MM buys 100 shares of a stock from one trader at a price of q, and then immediately sells 100 shares of stock to another trader at a price of $q + \Delta$, the MM records a profit of 100Δ.

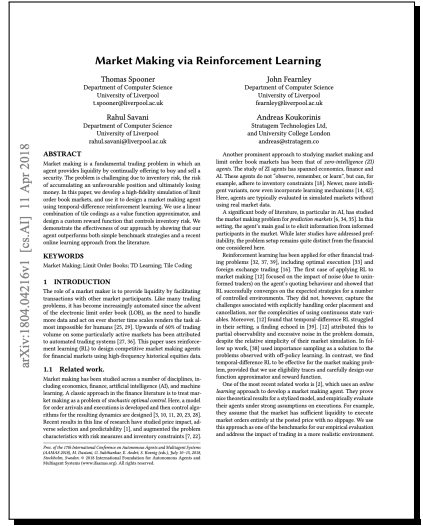
This describes the central goal of a profitable market making strategy: minimize the inventory risk of large movements in the price while simultaneously aiming to benefit from the bid-ask spread. The MM strategy has a *state*, which is the current inventory or holdings, receives as input order and price data, and must decide what quantities and at what prices to offer in the market. In the present paper we assume that the MM interacts with a *continuous double auction* via an *order book*, and the MM can place both *market and limit orders* to the order book.

A number of MM strategies have been proposed, and in many cases certain profit/loss guarantees have been given. But to the best of our knowledge all such guarantees (aside from [4]) have required

*Work performed while the author was in the CIS Department at the University of Pennsylvania and funded by a Simons Postdoctoral Fellowship

Key Papers: A Timeline

- Chan and Shelton (2001)
- Nevmyvaka, Feng and Kearns (2006)
- Avellaneda and Stoikov (2008)
- Abernethy and Kale (2013)
- Spooner et al. (2018)



Market Making via Reinforcement Learning

Thomas Spooner
Department of Computer Science
University of Liverpool
t.spooner@liverpool.ac.uk

John Fearnley
Department of Computer Science
University of Liverpool
fearnley@liverpool.ac.uk

Rahul Savani
Department of Computer Science
University of Liverpool
rahul.savani@liverpool.ac.uk

Andreas Koukorinis
Stratagem Technologies Ltd,
and University College London
andreas@stratagem.co

ABSTRACT

Market making is a fundamental trading problem in which an agent provides liquidity by continually offering to buy and sell a security. The problem is challenging due to inventory risk, the risk of accumulating an unfavourable position and ultimately losing money. In this paper, we develop a high-fidelity simulation of limit order book markets, and use it to design a market making agent using temporal-difference reinforcement learning. We use a linear combination of tile codings as a value function approximator, and design a custom reward function that controls inventory risk. We demonstrate the effectiveness of our approach by showing that our agent outperforms both simple benchmark strategies and a recent online learning approach from the literature.

KEYWORDS

Market Making; Limit Order Books; TD Learning; Tile Coding

1 INTRODUCTION

The role of a market maker is to provide liquidity by facilitating transactions with other market participants. Like many trading problems, it has become increasingly automated since the advent of the electronic limit order book (LOB), as the need to handle more data and act on ever shorter time scales renders the task almost impossible for humans [25, 29]. Upwards of 60% of trading volume on some particularly active markets has been attributed to automated trading systems [37, 56]. This paper uses reinforcement learning (RL) to design competitive market making agents for financial markets using high-frequency historical equities data.

1.1 Related work.

Market making has been studied across a number of disciplines, including economics, finance, artificial intelligence (AI), and machine learning. A classic approach in the finance literature is to treat market making as a problem of *stochastic optimal control*. Here, a model for order arrivals and executions is developed and then control algorithms for the resulting dynamics are designed [5, 31, 11, 20, 23, 28]. Recent results in this line of research have studied price impact, adverse selection and predictability [1], and augmented the problem characteristics with risk measures and inventory constraints [7, 22].

Another prominent approach to studying market making and limit order book markets has been that of *zero-intelligence* (ZI) agents. The study of ZI agents has spanned economics, finance and AI. These agents do not "observe, remember, or learn", but can, for example, adhere to inventory constraints [18]. Newer, more intelligent variants, now even incorporate learning mechanisms [14, 42]. Here, agents are typically evaluated in simulated markets without using real market data.

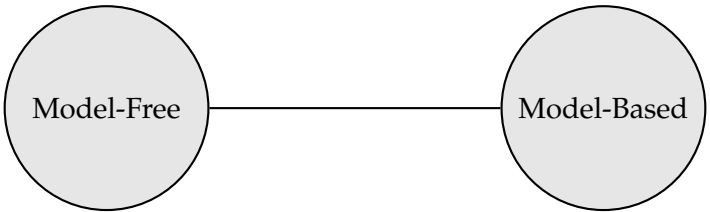
A significant body of literature, in particular in AI, has studied the *market making problem for prediction markets* [6, 34, 35]. In this setting, the agent's main goal is to elicit information from informed participants in the market. While later studies have addressed profitability, the problem setup remains quite distinct from the financial one considered here.

Reinforcement learning has been applied for other financial trading problems [32, 37, 38], including optimal execution [16] and foreign exchange trading [16]. The first case of applying RL to market making [13] focused on the impact of noise (due to uninformed traders) on the agent's quoting behaviour and showed that RL successfully converges on the expected strategies for a number of controlled environments. They did not, however, capture the challenges associated with explicitly handling order placement and cancellation, nor the complexities of using continuous state variables. Moreover, [13] found that temporal-difference RL struggled in their setting, a finding echoed in [39]. [12] attributed this to partial observability and excessive noise in the problem domain, despite the relative simplicity of their market simulation. In follow up work, [38] used importance sampling as a solution to the problems observed with off-policy learning. In contrast, we find temporal-difference RL to be effective for the market making problem, provided that we use eligibility traces and carefully design our function approximator and reward function.

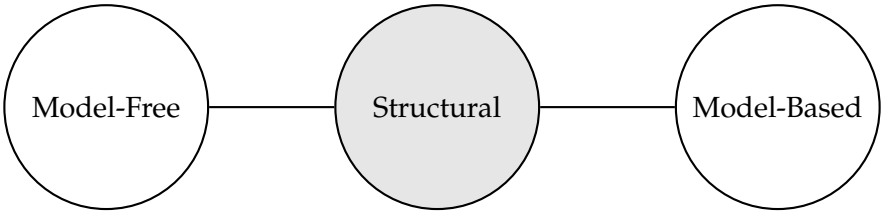
One of the most recent related works is [2], which uses an *online learning* approach to develop a market making agent. They prove nice theoretical results for a stylised model, and empirically evaluate their agents under strong assumptions on executions. For example, they assume that the market has sufficient liquidity to execute market orders entirely at the posted price with no slippage. We use this approach as one of the benchmarks for our empirical evaluation and address the impact of trading in a more realistic environment.

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), J. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

What is Structure?



What is Structure?



Reinforcement Learning

Algorithmic Trading

Leveraging Structure: *Gradient Factorisation*

Leveraging Structure: *Transition Acyclicity*

Reinforcement Learning

Markov Chains

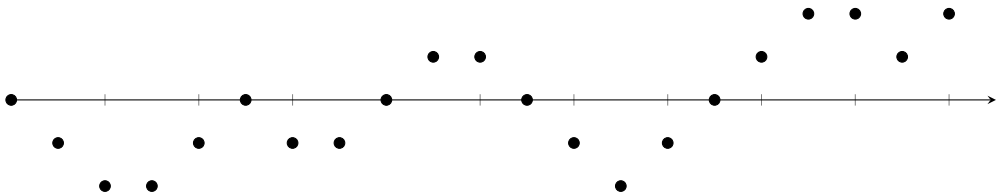
A *Markov chain* (MC) $\boldsymbol{X} = (X_0, X_1, \dots)$ is a discrete-time stochastic process.

The chain $\boldsymbol{X}$ occupies a *state-space* X s.t. $X_n \in \mathsf{X}$ for all $n \in \mathbb{N}$.

Markov Chains

A *Markov chain* (MC) $\boldsymbol{X} = (X_0, X_1, \dots)$ is a discrete-time stochastic process.

The chain $\boldsymbol{X}$ occupies a *state-space* X s.t. $X_n \in \mathsf{X}$ for all $n \in \mathbb{N}$.

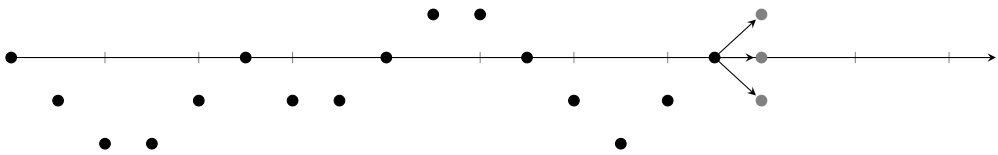


Markov Chains: Dynamics

A chain behaves according to a *transition kernel*:

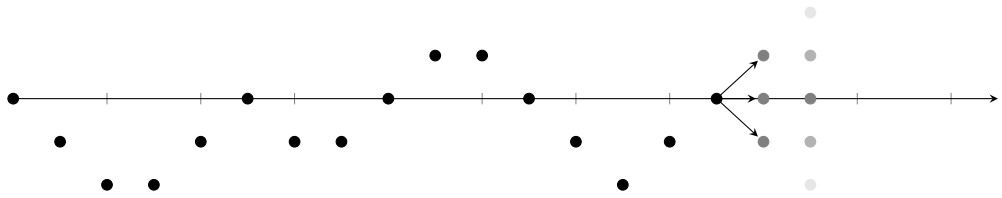
$$\boldsymbol{P} \doteq \{P(x, B) : x \in \mathsf{X}, B \in \mathfrak{B}(\mathsf{X})\},$$

where $P(x, B) \doteq \mathbb{P}(X_{n+1} \in B \,|\, X_n = x)$.



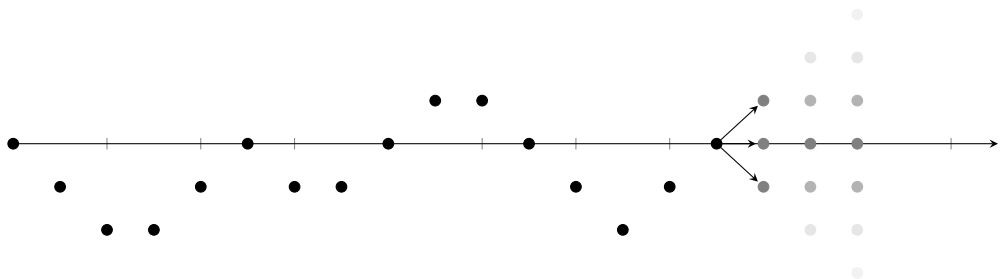
Markov Chains: Dynamics

We can naturally extend to *multi-step kernels*: $P_2(x, B) \doteq \int_{\mathsf{X}} P(x, \mathrm{d}y) \cdot P(y, B)$.



Markov Chains: Dynamics

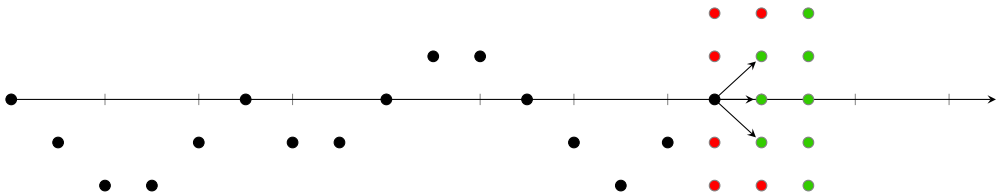
And the rest are defined inductively: $P_n(x, B) \doteq \int_{\mathsf{X}} P(x, \mathrm{d}y) \cdot P_{n-1}(y, B)$.



Markov Chains: Reachability

The n-step kernels describe the probability of *reaching* states in X.

A state $x \in \mathsf{X}$ is said to *lead to a set* $B \in \mathfrak{B}(\mathsf{X})$ if $P_n(x, B) > 0$ for some $n > 0$.



Markov Chains: Reachability

What about the probability of *ever returning to a set* $B \in \mathfrak{B}(\mathsf{X})$?

First Return

The time of first return to $B \in \mathfrak{B}(\mathsf{X})$ is

$$\tau_n(B) \doteq \min\{m \geqslant 1 \,:\, X_{n+m} \in B\}\,.$$

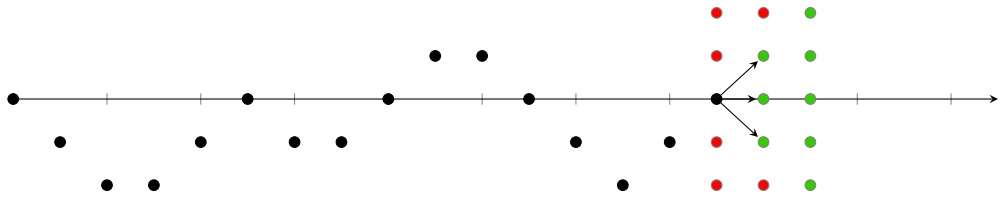
Reachability

The probability of ever reaching B is then

$$P_\infty(x, B) \doteq \mathbb{P}(\tau_n(B) < \infty \,|\, X_n = x)\,.$$

Markov Chains: Reachability

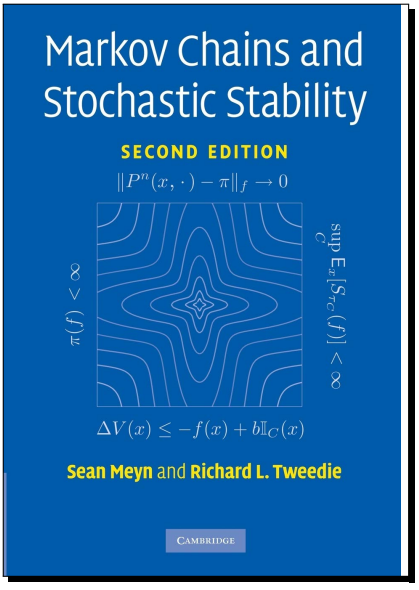
If $P_\infty(x, B) > 0$ then there exists a path from $x \in \mathsf{X}$ to $B \in \mathfrak{B}(\mathsf{X})$.



Markov Chains

For further reading:

- Meyn and Tweedie (2012).
- Grimmett and Stirzaker (2006).



Markov Decision Processes

Markov decision processes (MDPs) extend MCs by the inclusion of:

1. an action space U; and
2. a reward function $r : \mathsf{X} \times \mathsf{U} \times \mathsf{X} \to \mathbb{R}$.

The former leads us to define a *joint state-action process* $(\boldsymbol{X}, \boldsymbol{U})$, where

$$P_1^\pi(x, B) \doteq \int_{\mathsf{U}(x)} P_1(x, B \mid u) \cdot \mathrm{d}F_\pi(u \mid x) \,.$$

Markov Decision Processes: Policies

The *behaviour* of an agent is described by a *policy* $\pi \in \Pi$, where

$$\Pi \doteq \{\mathsf{X} \to \mathcal{P}(\mathsf{U})\}.$$

Deterministic Policies

"If price is higher than c, then buy…"

$$\pi(u \mid x; c) = \delta_u(\mathbf{1}_{x \geqslant c})$$

Stochastic Policies

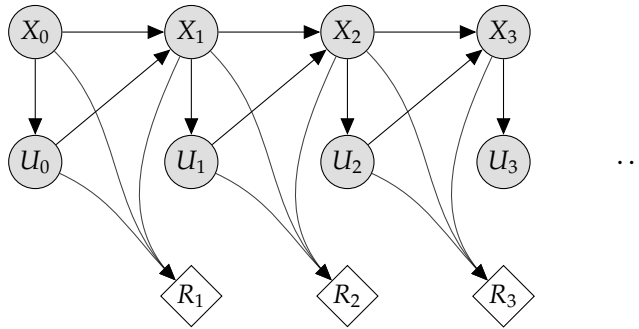
"Quote prices about centre $f_\theta(x)$…"

$$\pi(u \mid x; \theta) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{(f_\theta(x)-x)^2}{2}}$$

Markov Decision Processes: Dynamics

The process $(\boldsymbol{X}, \boldsymbol{U})$ generates a trajectory according to:

$$
\begin{aligned}
X_0 &\sim d_0(\cdot), \\
U_0 &\sim \pi(\cdot \mid X_0), \\
X_1 &\sim P_1^{\pi}(X_0, \cdot), \\
R_1 &= r(X_0, U_0, X_1), \\
U_1 &\sim \pi(\cdot \mid X_1), \\
X_2 &\sim P_1^{\pi}(X_1, \cdot), \\
R_2 &= r(X_1, U_1, X_2), \\
&\vdots
\end{aligned}
$$



Markov Decision Processes: Performance Criteria

How do we measure the performance of a policy $\pi \in \Pi$?

$$G_n^\gamma \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{n+1} + \gamma G_{n+1}.$$

Undiscounted Setting
Let $\gamma \to 1$ and define

$$J(\pi) = \mathbb{E}_\pi\left[G_0^1 \,\middle|\, X_0 \sim d_0\right]$$

Discounted Setting
Let $\gamma \in [0, 1)$ and define

$$J(\pi) = \mathbb{E}_\pi\left[G_0^\gamma \,\middle|\, X_0 \sim d_0\right]$$

Markov Decision Processes: Value Functions

We can now measure the *value of a state/state-action pair* under a policy $\pi \in \Pi$.

- Again, we appeal to the expected return…

State-Value

$$v_\pi^\gamma(x) \doteq \mathbb{E}_\pi\left[G_n^\gamma \,\middle|\, X_n = x\right]$$

Action-Value

$$q_\pi^\gamma(x,u) \doteq \mathbb{E}_\pi\left[G_n^\gamma \,\middle|\, X_n = x, U_n = u\right]$$

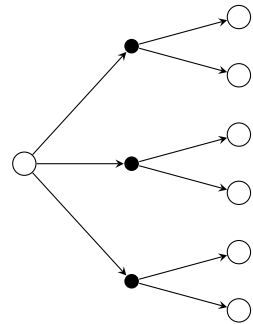
Markov Decision Processes: Bellman Equations

The value functions can be expressed recursively:

- This is the essence of *dynamic programming*.

Backup diagrams provide some insight:

- White nodes represent states.
- Black nodes represent actions.
- Edges represent probabilistic relations (transitions).

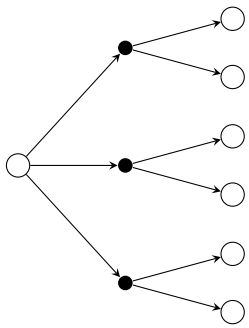


Markov Decision Processes: Bellman Equations

First define the *expected immediate reward*: $r(x,u) \doteq \mathbb{E}[R_{n+1} \,|\, X_n = x, U_n = u]$.

It then follows that:

$$
\begin{aligned}
q_\pi^\gamma(x,u) &= \mathbb{E}_\pi\left[G_n^\gamma \,\middle|\, X_n = x, U_n = u\right], \\
&= \mathbb{E}_\pi\left[R_{n+1} + \gamma G_{n+1}^\gamma \,\middle|\, X_n = x, U_n = u\right], \\
&= r(x,u) + \gamma \mathbb{E}_\pi\left[G_{n+1}^\gamma \,\middle|\, X_n = x, U_n = u\right], \\
&= \underbrace{r(x,u)}_{\text{Immediate}} + \underbrace{\gamma \mathbb{E}_\pi[v_\pi^\gamma(X_{n+1}) \,|\, X_n = x, U_n = u]}_{\text{Future}}.
\end{aligned}
$$



Markov Decision Processes: Optimality

Our objective can be expressed as: $J(\pi) = \mathbb{E}_\pi\left[v_\pi^\gamma(X_0) \,\middle|\, X_0 \sim d_0\right]$.

Value functions thus define a *partial ordering over the set of policies* Π:

$$\pi \succeq \pi' \iff v_\pi^\gamma(x) \geqslant v_{\pi'}^\gamma(x) \quad \forall\, x \in \mathsf{X}.$$

Optimal Policy

$$\pi_\star \in \operatorname*{arg\,max}_{\pi \in \Pi} J(\pi)$$

Optimal Value

$$v_\star^\gamma(x) \doteq \max_{\pi \in \Pi} v_\pi^\gamma(x) = v_{\pi_\star}^\gamma(x)$$

Markov Decision Processes: Optimality

What can we say about the set $\Pi_\star \doteq \arg\max_{\pi \in \Pi} J(\pi)$?

Existence

If values are bounded,[1] then $|\Pi_\star| > 0$.

Uniqueness

No guarantees in general!

[1] *Assured, for example, if* $r : \mathsf{X} \times \mathsf{U} \times \mathsf{X} \to \mathbb{R}$ *is bounded and* $\gamma \in [0, 1)$.

Markov Decision Processes: Optimality

The set $\Pi_\star$ also induces an optimal action-value function:

$$q_\star^\gamma(x,u) \doteq \max_{\pi\in\Pi} q_\pi(x,u) = q_{\pi_\star}(x,u) \quad \forall\, \pi_\star \in \Pi_\star.$$

The functions $v_\star^\gamma$ and $q_\star^\gamma$ are then related by the equation,

$$v_\star^\gamma(x) = \max_{u\in U} q_\star^\gamma(x,u),$$

yielding the *Bellman Optimality equation*:

$$\begin{aligned} q_\star^\gamma(x,u) &= r(x,u) + \gamma\mathbb{E}[v_\star^\gamma(X_{n+1}) \,|\, X_n = x, U_n = u]\,, \\ &= r(x,u) + \gamma\mathbb{E}\left[\max_{u'\in U} q_\star^\gamma(X_{n+1},u') \,\middle|\, X_n = x, U_n = u\right]. \end{aligned}$$

Policy Evaluation

The policy evaluation problem[2] is stated as the following:

Given an MDP and a fixed policy $\pi \in \Pi$, compute the value functions $v_\pi^\gamma(x)$ and $q_\pi^\gamma(x, u)$ for all $x \in \mathsf{X}$ and $u \in \mathsf{U}$.

[2] *Often referred to as the prediction problem.*

Policy Evaluation

The "simplest" evaluation algorithm is **value iteration** (VI).[3]

- VI converges for fixed $\pi \in \Pi$:

$$\lim_{i \to \infty} \hat{v}_i \to v_\pi.$$

- Practically…it's not so good.
- Requires *full backups*.
- Uses *lots of memory*.

```
1  Initialise i ← 0, v̂_i, q̂_i arbitrarily;
2  while ||v̂_i − v_π||_∞ > ε do
3      v̂_{i+1} ← v̂_i;
4      q̂_{i+1} ← q̂_i;
5      for x ∈ X do
6          for u ∈ U do
7              q̂_{i+1}(x, u) ← r(x, u) +
                 γ Σ_{x'∈X} P(x, {x'} | u) v̂_i(x');
8          end
9          v̂_{i+1}(x) = Σ_{u∈U} π(u | x) q̂_{i+1}(x, u);
10     end
11     i ← i + 1;
12 end
```

[3] *Technically this isn't historically referred to as value iteration, but it's pretty close…*

Policy Evaluation

What if we don't have direct access to P or r?

Policy Evaluation

What if we don't have direct access to P or r?

Sampling to the rescue!

Policy Evaluation

Temporal-difference (TD) learning uses *bootstrapping* to perform *sample backups*.

- TD converges for fixed $\pi \in \Pi$:

$$\lim_{i \to \infty} \hat{v}_i \to v_\pi.$$

- Often more practical than VI.
- *Variance is higher*.
- Sensitive to *sampling distribution*.

Sample $x \sim d_0$;
Initialise $i \leftarrow 0$, and $\hat{v}$ arbitrarily;
loop
    Sample action: $u \sim \pi(\cdot|x)$;
    Transition to new state: $x' \sim P(x, \cdot)$;
    Compute the temporal-difference error:
    $\delta \leftarrow r(x, u, x') + \gamma \hat{v}(x') - \hat{v}(x)$;
    Update estimate: $\hat{v}(x) \leftarrow \hat{v}(x) + \alpha\delta$;
    Innovate: $x \leftarrow x'$;
end

Policy Evaluation: Approximate Methods

What now if U is infinite? Or if no exact representation of v_π exists?

Policy Evaluation: Approximate Methods

What now if U is infinite? Or if no exact representation of v_π exists?

(Stochastic) Gradient Descent to the rescue, as ever!

Policy Evaluation: Approximate Methods

Approximate methods resort to parameterised representations:

$$v_\pi(\cdot) \approx \hat{v}(\cdot; \boldsymbol{w}).$$

An important class is the set of *linear approximations*:

$$\mathcal{F}_{\boldsymbol{\phi}} = \left\{ f(\cdot) = \langle \boldsymbol{w}, \boldsymbol{\phi}(\cdot) \rangle = \sum_{i=1}^{n} w_i \phi_i(\cdot) \;\middle|\; \boldsymbol{w} \in \mathbb{R}^n \right\},$$

for which

$$\nabla_{\boldsymbol{w}} f(\cdot; \boldsymbol{w}) = \boldsymbol{\phi}(\cdot) \quad \forall f \in \mathcal{F}_{\boldsymbol{\phi}}.$$

Policy Evaluation: Approximate Methods

There are lots of different choices for $\boldsymbol{\phi}$... get creative!

Polynomial

$$\phi_i(\boldsymbol{x}) \doteq \prod_{j=1}^{n} x_j^{e_{ij}}$$

Fourier

$$\phi_i(\boldsymbol{x}) \doteq \cos(\pi \cdot \langle \boldsymbol{c}_i, \boldsymbol{x} \rangle)$$

Tile Coding

$$\phi_i(\boldsymbol{x}) \doteq \frac{1}{K} \sum_{j=1}^{K} \mathbf{1}\Big(\mathsf{Project}_j[\boldsymbol{x}] = i\Big)$$

Policy Evaluation: Approximate Methods

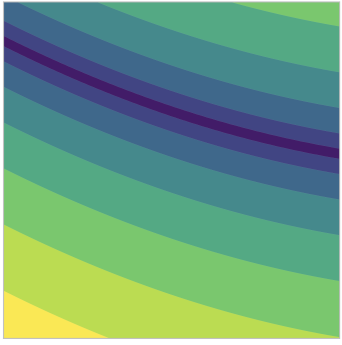


Figure: Ground truth.

$\approx$

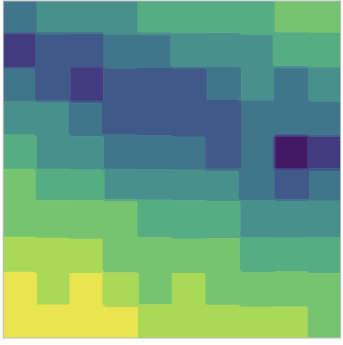


Figure: Tile Coding ($K = 1$)

Policy Evaluation: Approximate Methods

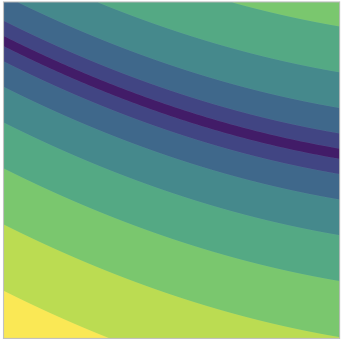


Figure: Ground truth.

$\approx$

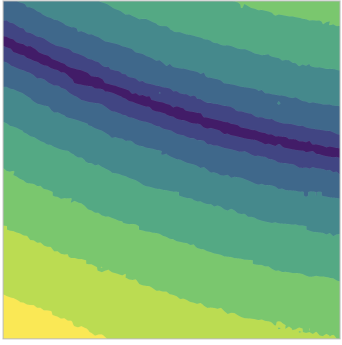


Figure: Tile Coding ($K = 16$)

Policy Evaluation: Approximate Methods

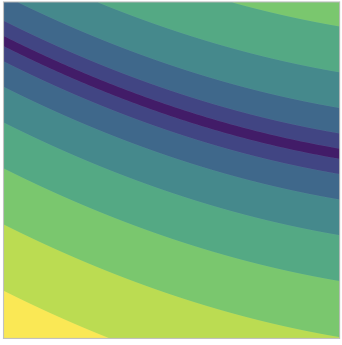


Figure: Ground truth.

$\approx$

Figure: Polynomial (3^{rd}-order)

Policy Evaluation: Approximate Methods

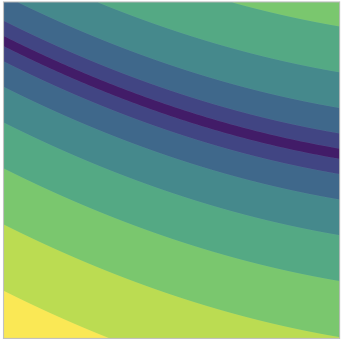


Figure: Ground truth.

$\approx$

Figure: Fourier (7^{th}-order)

Policy Evaluation: Approximate Methods

Let us define an *(mean-squared) error* associated with the estimate $\hat{v}_{\boldsymbol{w}}$,

$$\mathcal{E}(\boldsymbol{w}) \doteq \|\hat{v}_{\boldsymbol{w}} - v_\pi\|_d^2 = \mathbb{E}\left[[\hat{v}_{\boldsymbol{w}}(X) - v_\pi(X)]^2 \,\middle|\, X \sim d(\cdot) \right],$$

and then differentiate to give:

$$\nabla_{\boldsymbol{w}} \mathcal{E}(\boldsymbol{w}) = 2\mathbb{E}[[\hat{v}_{\boldsymbol{w}}(X) - v_\pi(X)]\, \nabla_{\boldsymbol{w}} \hat{v}_{\boldsymbol{w}}(X) \,|\, X \sim d(\cdot)]\,.$$

Policy Evaluation: Approximate Methods

The corresponding *stochastic approximation*[4] is given by

$$\boldsymbol{w}_{n+1} \leftarrow \boldsymbol{w}_n - \alpha_t[\hat{v}_{\boldsymbol{w}}(X_n) - \underbrace{v_\pi(X_n)}_{\text{Chicken} \iff \text{Egg?}}]\nabla_{\boldsymbol{w}}\hat{v}_{\boldsymbol{w}}(X).$$

Fortunately, we can replace $v_\pi(X_n)$ with any *unbiased estimator* Z_n:

MC Let $Z_n \doteq \sum_{i=1}^{N-n} \gamma^{i-1} R_{n+i}$.

TD Let $Z_n \doteq R_{n+1} + \gamma\hat{v}(X_{n+1})$.

[4] *Vivek S Borkar and Sean P Meyn. "The ODE Method for Convergence of Stochastic Approximation and Reinforcement Learning". In: SIAM Journal on Control and Optimization 38.2 (2000), pp. 447–469.*

Policy Evaluation

There have been many extensions to these methods:[5]

Least-Squares Methods[6]

Deep Learning

True-Online Methods[7]

Eligibility Traces[8]

Gradient Methods[9]

[5] *Adam White and Martha White.* "Investigating Practical Linear Temporal Difference Learning". *In: arXiv preprint arXiv:1602.08771 (2016).*

[6] *Steven J Bradtke and Andrew G Barto.* "Linear Least-Squares Algorithms for Temporal Difference Learning". *In: Machine Learning 22.1 (1996), pp. 33–57*

[7] *Harm Van Seijen et al.* "True Online Temporal-Difference Learning". *In: JMLR 17.1 (2016), pp. 5057–5096*

[8] *Hamid Reza Maei.* "Gradient Temporal-Difference Learning Algorithms". *2011*

[9] *Richard S Sutton and Andrew G Barto.* Reinforcement Learning: An Introduction. *MIT press, 2018*

Policy Optimisation

The policy optimisation problem[10] is stated as the following:

Given an MDP, find an optimal policy $\pi_\star \in \Pi_\star$.

[10] *Often referred to as the control problem.*

Policy Optimisation: Value-Based Methods

The first class of algorithms are *value-based methods*.

The goal is to construct a sequence $(\hat{q}_0, \hat{q}_1, \hat{q}_2, \dots)$ such that

$$\lim_{n \to \infty} q_n = q_\star.$$

Since acting greedily wrt. $q_\star$ is optimal, it follows that

$$\lim_{n \to \infty} \mathsf{Greedy}[q_n] \in \Pi_\star.$$

Policy Optimisation: Value-Based Methods

We now use the **VI** algorithm from before, but for control.

- Note the use of the Bellman optimality equations!

Initialise $i \leftarrow 0$, $\hat{v}_i(x)$, $\hat{q}_i(x,u)$ arbitrarily;
while $||\hat{v}_i - v_*||_\infty > \varepsilon$ **do**
  $\hat{v}_{i+1} \leftarrow \hat{v}_i$;
  $\hat{q}_{i+1} \leftarrow \hat{q}_i$;
  for $x \in \mathsf{X}$ **do**
    for $u \in \mathsf{U}$ **do**
      $\hat{q}_{i+1}(x,u) \leftarrow r(x,u) + \gamma \sum_{x' \in \mathsf{X}} P(x, \{x'\} \mid u) \max_{u' \in \mathsf{U}} \hat{q}_i(x', u')$;
    end
    $\hat{v}_{i+1}(x) = \max_{u \in \mathsf{U}} \hat{q}_{i+1}(x,u)$;
  end
  $i \leftarrow i + 1$;
end

Policy Optimisation: Value-Based Methods

As with evaluation, we can *leverage sampling* when P or r are not available.

- The direct result is **Watkins' Q-Learning** algorithm.

---

1 Initialise $x \sim d_0$, and $\hat{q}_w(x, u)$ arbitrarily;

2 **loop**

3 Sample action: $u \sim \pi(\cdot|x)$;

4 Transition to new state: $x' \sim P(x, \cdot)$;

5 Compute the temporal-difference error: $\delta \leftarrow r(x, u, x') + \gamma \max_{u' \in \mathsf{U}} \hat{q}(x', u') - \hat{q}(x, u)$;

6 Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\delta\nabla_{\boldsymbol{w}}\hat{q}_{\boldsymbol{w}}(x, u)$;

7 Innovate: $x \leftarrow x'$;

8 **end**

Policy Optimisation: Policy-Gradient Methods

The second class of algorithms are *policy-gradient methods*.

Our goal here is to construct a sequence $(\theta_0, \theta_1, \theta_2, \dots)$ such that

$$\lim_{n\to\infty} \nabla_{\theta_n} J(\pi_{\theta_n}) = 0,$$

whereby, we hope,[11]

$$\lim_{n\to\infty} \pi_{\theta_n} \in \Pi_\star.$$

[11] *This is not guaranteed!*

Policy Optimisation: Policy-Gradient Methods

Let $d_\pi^\gamma \in \Delta(\mathsf{X})$ denote the *(improper) discounted occupancy measure*.

- This describes "how long we spend in each state (with discounting)."

The *policy gradient theorem*[12] then tells us that

$$\nabla_\theta J(\pi_\theta) = \underbrace{\mathbb{E}_{d_{\pi_\theta}^\gamma, \pi_\theta}[q_{\pi_\theta}(X, U) \nabla_\theta \log \pi_\theta(U \mid X)]}_{\text{Score-Based Gradient Estimator}},$$

which allows us to perform local search:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta).$$

[12] Richard S Sutton et al. *"Policy Gradient Methods for Reinforcement Learning with Function Approximation"*. In: Proc. of NeurIPS 12 (1999).

Policy Optimisation: Policy-Gradient Methods

There are lots of variants on policy gradient methods:

- Most revolve around how to reduce variance on estimates of $\nabla_{\theta} J(\pi_{\theta})$.

Baselines **Actor-Critic Methods**

Generalised Advantage Estimation[13]

SVRPG[14] **Topological**[15]

[13] *John Schulman et al. "High-Dimensional Continuous Control using Generalized Advantage Estimation". In: arXiv preprint arXiv:1506.02438 (2015)*

[14] *Matteo Papini et al. "Stochastic Variance-Reduced Policy Gradient". In: Proc. of ICML. 2018, pp. 4026–4035*

[15] *Yasuhiro Fujita and Shin-ichi Maeda. "Clipped Action Policy Gradient". In: Proc. of ICML. 2018, pp. 1597–1606*

Policy Optimisation: Policy-Gradient Methods

The *baseline trick* is particularly important (and will feature later):

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{d^\gamma_{\pi_\theta}, \pi_\theta}\left[\left[q_{\pi_\theta}(X, U) - b(X)\right] \nabla_\theta \log \pi_\theta(U \mid X)\right],$$

which holds because

$$\mathbb{E}_{d^\gamma_{\pi_\theta}, \pi_\theta}\left[b(X) \nabla_\theta \log \pi_\theta(U \mid X)\right] = 0.$$

The *optimal baseline is impractical* to derive or compute.

- Instead, we typically use $b(x) \doteq v_{\pi_\theta}(x)$.
- But there are better[16] choices...

[16] *If it wasn't already clear, this is a hint for later!*

Algorithmic Trading

Trading Formalism

The trading universe comprises *one riskless asset (cash) and* $m > 0$ *risky assets*.

Trading Formalism

The trading universe comprises *one riskless asset (cash) and* $m > 0$ *risky assets*.

Riskless Asset

- Price fixed at unity.
- Agent holdings are denoted by

$$\boldsymbol{C} \doteq (C_0, C_1, C_2, \ldots) .$$

Trading Formalism

The trading universe comprises *one riskless asset (cash) and* $m > 0$ *risky assets*.

Riskless Asset

- Price fixed at unity.
- Agent holdings are denoted by

$$\boldsymbol{C} \doteq (C_0, C_1, C_2, \ldots)\,.$$

Risky Assets

- Prices $\boldsymbol{Z}$ evolve stochastically with

$$\boldsymbol{Z}_n = [(\boldsymbol{Z}_n)_1, (\boldsymbol{Z}_n)_2, \ldots, (\boldsymbol{Z}_n)_m] \in \mathbb{R}^m.$$

- Agent holdings are $\boldsymbol{\Omega}$ with

$$\boldsymbol{\Omega}_n = [(\boldsymbol{\Omega}_n)_1, (\boldsymbol{\Omega}_n)_2, \ldots, (\boldsymbol{\Omega}_n)_m] \in \mathbb{R}^m.$$

Trading Formalism

The agent portfolio is denoted by the tuple $\Lambda_n \doteq (C_n, \mathbf{\Omega}_n)$ for all $n \geqslant 0$.

The *mark-to-market value* of Λ_n is the inner product:

$$\Upsilon_n \doteq C_n + \langle \mathbf{\Omega}_n, \mathbf{Z}_n \rangle \,.$$

Trading Formalism

The agent portfolio is denoted by the tuple $\Lambda_n \doteq (C_n, \mathbf{\Omega}_n)$ for all $n \geqslant 0$.

The *mark-to-market value* of Λ_n is the inner product:

$$\Upsilon_n \doteq C_n + \langle \mathbf{\Omega}_n, \mathbf{Z}_n \rangle .$$

We can then compute the discrete derivative as follows:

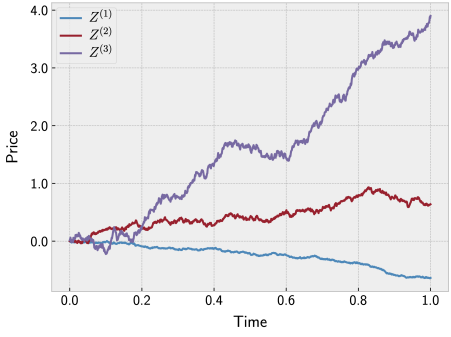
$$\begin{aligned}
\Delta\Upsilon_n &= \Delta C_n + \Delta \langle \mathbf{\Omega}_n, \mathbf{Z}_n \rangle , \\
&= \Delta C_n + \langle \Delta\mathbf{\Omega}_n, \mathbf{Z}_n \rangle + \langle \Delta\mathbf{\Omega}_n, \Delta\mathbf{Z}_n \rangle + \langle \mathbf{\Omega}_n, \Delta\mathbf{Z}_n \rangle , \\
&= \underbrace{\Delta C_n + \langle \Delta\mathbf{\Omega}_n, \mathbf{Z}_n \rangle}_{\text{Trading}} + \overbrace{\langle \mathbf{\Omega}_{n+1}, \Delta\mathbf{Z}_n \rangle}^{\text{Note the } n+1!}_{\text{Speculation}} .
\end{aligned}$$

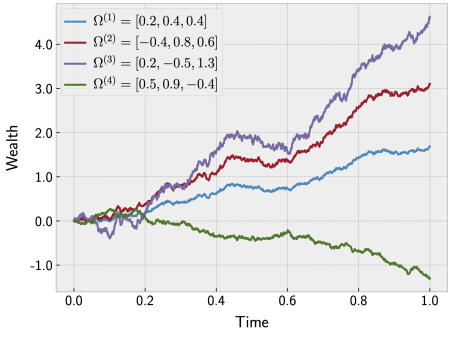
As an example, consider a portfolio consisting of three risky assets:

$$\Delta \boldsymbol{Z}_n = \begin{bmatrix} -0.5 \\ 0.5(1.5 - Z_n^{(2)}) \\ 3 \end{bmatrix} \Delta t + \begin{bmatrix} 5 \\ 15 \\ 25 \end{bmatrix} \circ \boldsymbol{N}_n.$$

As an example, consider a portfolio consisting of three risky assets:

$$\Delta \boldsymbol{Z}_n = \begin{bmatrix} -0.5 \\ 0.5(1.5 - Z_n^{(2)}) \\ 3 \end{bmatrix} \Delta t + \begin{bmatrix} 5 \\ 15 \\ 25 \end{bmatrix} \circ \boldsymbol{N}_n.$$





Trading Formalism: Transactions

Of course, we can do more than just buy-and-hold!

- So lets enrich our model to include transactions:

Sell
Let $\boldsymbol{\nu}^- \geqslant 0$ denote the agent's *ask transaction flow.*

Buy
Let $\boldsymbol{\nu}^+ \geqslant 0$ denote the agent's *bid transaction flow.*

Trading Formalism: Transactions

Of course, we can do more than just buy-and-hold!

- So lets enrich our model to include transactions:

Sell
Let $\boldsymbol{\nu}^- \geqslant 0$ denote the agent's *ask transaction flow.*

Buy
Let $\boldsymbol{\nu}^+ \geqslant 0$ denote the agent's *bid transaction flow.*

We can now express the inventory process:

$$\Delta \boldsymbol{\Omega}_n = \boldsymbol{\nu}_n^+ - \boldsymbol{\nu}_n^- \quad \Longrightarrow \quad \boldsymbol{\Omega}_n = \omega_0 + \sum_{k=0}^{n-1} \boldsymbol{\nu}_k^+ - \boldsymbol{\nu}_k^-$$

Trading Formalism: Transactions

Of course, we can do more than just buy-and-hold!

- So lets enrich our model to include transactions:

Sell
Let $\boldsymbol{\nu}^- \geqslant 0$ denote the agent's *ask transaction flow.*

Buy
Let $\boldsymbol{\nu}^+ \geqslant 0$ denote the agent's *bid transaction flow.*

And its positive/negative parts:

$$\Delta\boldsymbol{\Omega}_n^{\pm} = \boldsymbol{\nu}_n^{\pm} \quad \Longrightarrow \quad \boldsymbol{\Omega}_n^{\pm} = \omega_0^{\pm} + \sum_{k=0}^{n-1} \boldsymbol{\nu}_k^{\pm} \quad \Longrightarrow \quad \boldsymbol{\Omega}_n = \boldsymbol{\Omega}_n^{+} - \boldsymbol{\Omega}_n^{-}$$

Trading Formalism: Transactions

So, what happens when the agent *buys k units of a risky asset*?

Agent

- Sends kZ_n cash to counterparty.
  - $C_{n+1} \leftarrow C_n - kZ_n$.
  - $\Omega_{n+1} \leftarrow \Omega_n + k$.
  - $\nu_n = \nu_n^+ = k$.

Counterparty

- Sends k units to agent.

Trading Formalism: Transactions

So, what happens when the agent *sells k units of a risky asset*?

Agent

- Sends k units to counterparty.
  - $C_{n+1} \leftarrow C_n + kZ_n$.
  - $\Omega_{n+1} \leftarrow \Omega_n - k$.
  - $\nu_n = -\nu_n^- = -k$.

Counterparty

- Sends kZ_n cash to agent.

Trading Formalism: Transactions

Is it realistic to assume that you can trade at $\boldsymbol{Z}_n$?

Trading Formalism: Transactions

Is it realistic to assume that you can trade at $\boldsymbol{Z}_n$? No!

- Latency.
- Slippage.
- Transaction fees.
- Information asymmetry.

Trading Formalism: Transactions

Is it realistic to assume that you can trade at $\boldsymbol{Z}_n$? No!

- Latency.
- Slippage.
- Transaction fees.
- Information asymmetry.

We can account for some effects with a *(temporary) impact function* $f : \cdots \to \mathbb{R}$

$$\Delta C_n \approx -\boldsymbol{Z}_n \cdot \boldsymbol{\nu}_n + f(\boldsymbol{Z}_n, \boldsymbol{\nu}_n^+, \boldsymbol{\nu}_n^-, \dots),$$

but reality is a little more complicated…

Limit Order Books

Many electronic markets operate as *limit order books (LOB)*.

A *limit order (LO)* is a triple:

$$o \doteq (n_o, \omega_o, z_o).$$

The LOB is then the collection

$$\mathcal{B}_n \doteq \mathcal{B}_n^+ \cup \mathcal{B}_n^-$$

where

$$\mathcal{B}_n^+ \doteq \{o : n_o \leqslant n, \omega_o > 0\},$$
$$\mathcal{B}_n^- \doteq \{o : n_o \leqslant n, \omega_o < 0\}.$$

	Price	Offers
	101.0	12
	100.5	13
	100.0	
35	99.5	
11	99.0	
3	98.5	
Bids		

Limit Order Books

Many electronic markets operate as *limit order books (LOB)*.

A market order is a special case.

- Let the price tend to 0 or ∞.

The process is called: *walking the book.*

- Transact until liquidity is exhausted.
- Or until the request is satisfied.

	Price	Offers
	101.0	12
	100.5	13
	100.0	
35	99.5	
11	99.0	
3	98.5	
Bids		

Limit Order Books

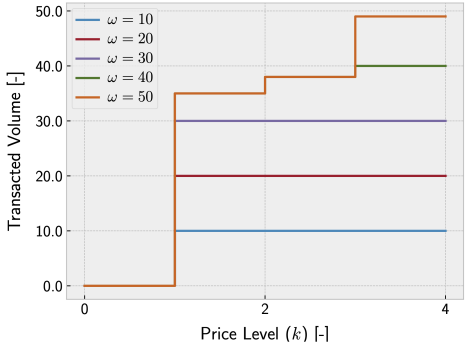
Many electronic markets operate as *limit order books (LOB)*.

A market order (MO) is a special case.

- Let the price tend to 0 or ∞.

The process is called: *walking the book.*

- Transact until liquidity is exhausted.
- Or until the request is satisfied.



Limit Order Books

Why do you favour LOs over MOs?

Limit Order

- Passive execution.
- Better price.

Market Order

- Immediate execution.
- Incurs slippage.

Market Making

Market making is a canonical problem in algorithmic trading.

- Market makers (MMs) quote bid and ask prices.
- They provide liquidity to a market.
- Derive profit of transactions.

Market Making

Market making is a canonical problem in algorithmic trading.

- Market makers (MMs) quote bid and ask prices.
- They provide liquidity to a market.
- Derive profit of transactions.

There are some significant challenges for MMs:

- Balancing transaction rate and spread.
  - *Optimal Pricing*
- Avoiding exposure to large market movements.
  - *Inventory Risk*

Market Making: Avellaneda-Stoikov Model

The Avellaneda-Stoikov model is a parametric approximation of an LOB.

Agent trades in *unit LOs with prices about the mid-price*: $Z_n^{\pm} = Z_n \mp \delta_n^{\pm}$.

Market Making: Avellaneda-Stoikov Model

The Avellaneda-Stoikov model is a parametric approximation of an LOB.

Agent trades in *unit LOs with prices about the mid-price*: $Z_n^{\pm} = Z_n \mp \delta_n^{\pm}$.

Prices

Evolve as a random walk.

$$Z_0 = z_0,$$
$$Z_n = Z_{n-1} + \sigma N_n.$$

Market Making: Avellaneda-Stoikov Model

The Avellaneda-Stoikov model is a parametric approximation of an LOB.

Agent trades in *unit LOs with prices about the mid-price*: $Z_n^{\pm} = Z_n \mp \delta_n^{\pm}$.

Prices

Evolve as a random walk.

$$Z_0 = z_0,$$
$$Z_n = Z_{n-1} + \sigma N_n.$$

Transactions

Modelled as Poisson processes:

$$\lambda^{\pm}(\delta^{\pm}) = A^{\pm} \exp(-k^{\pm}\delta^{\pm}).$$

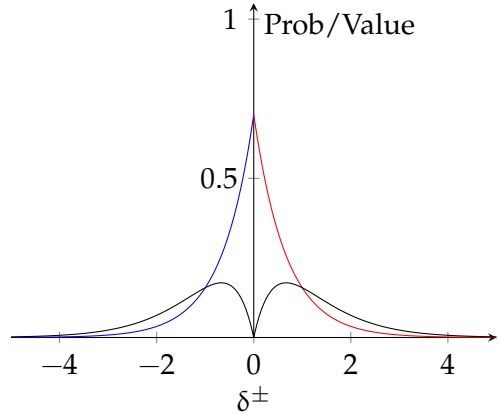
Market Making: Avellaneda-Stoikov Model

The probability of transacting decreases as δ^+, δ^- grow.

The *expected value of transacting is (quasi) concave*!

- There is an "optimal" price.

But we also have to worry about inventory…



Market Making: (Quadratic) MDP Formulation

We begin by defining states/actions:

$$X_n \doteq (n, Z_n, \Omega_n^+, \Omega_n^-) \qquad \text{and} \qquad U_n \doteq (\delta_n^+, \delta_n^-).$$

Market Making: (Quadratic) MDP Formulation

We begin by defining states/actions:

$$X_n \doteq (n, Z_n, \Omega_n^+, \Omega_n^-) \qquad \text{and} \qquad U_n \doteq (\delta_n^+, \delta_n^-).$$

Letting $x_0 \doteq (0, z_0, 0, 0)$, we define the objective as

$$\begin{aligned} J(\pi) &= \mathbb{E}_\pi \left[\Upsilon_N - \beta \sum_{n=0}^{N-1} \Omega_{n+1}^2 \Delta Z_n \,\middle|\, X_0 = x_0 \right], \\ &= \mathbb{E}_\pi \left[\sum_{n=0}^{N-1} \Delta C_n + Z_n \Delta \Omega_n + \Omega_{n+1} \Delta Z_n \left(1 - \beta \Omega_{n+1}\right) \,\middle|\, X_0 = x_0 \right]. \end{aligned}$$

Market Making: (Quadratic) MDP Formulation

Observe that our cash process must evolve according to

$$\begin{aligned}\Delta C_n &= (Z_n + \delta_n^-)\Delta\Omega_n^- - (Z_n - \delta_n^+)\Delta\Omega_n^+, \\ &= \delta_n^-\Delta\Omega_n^- + \delta_n^+\Delta\Omega_n^+ - Z_n\Delta\Omega_n.\end{aligned}$$

Market Making: (Quadratic) MDP Formulation

Observe that our cash process must evolve according to

$$\begin{aligned}\Delta C_n &= (Z_n + \delta_n^-)\Delta\Omega_n^- - (Z_n - \delta_n^+)\Delta\Omega_n^+, \\ &= \delta_n^-\Delta\Omega_n^- + \delta_n^+\Delta\Omega_n^+ - Z_n\Delta\Omega_n.\end{aligned}$$

We can thus simplify the objective to give:

$$J(\pi) = \mathbb{E}_\pi\left[\sum_{n=0}^{N-1} \delta_n^-\Delta\Omega_n^- + \delta_n^+\Delta\Omega_n^+ + \Omega_{n+1}\Delta Z_n\,(1 - \beta\Omega_{n+1}) \,\middle|\, X_0 = x_0\right].$$

Market Making: (Quadratic) MDP Formulation

One possible instantiation of the reward is thus

$$R_{n+1} = \delta_n^- \Delta\Omega_n^- + \delta_n^+ \Delta\Omega_n^+ + \Omega_{n+1}\Delta Z_n(1 - \beta\Omega_{n+1}).$$

From this it follows that

$$J(\pi) = \underbrace{\mathbb{E}_\pi\left[\sum_{n=0}^{N-1} R_{n+1} \,\middle|\, X_0 = x_0\right]}_{\text{Look familiar?!}}$$

Market Making: (Quadratic) MDP Formulation

One possible instantiation of the reward is thus

$$R_{n+1} = \delta_n^- \Delta\Omega_n^- + \delta_n^+ \Delta\Omega_n^+ + \Omega_{n+1}\Delta Z_n(1 - \beta\Omega_{n+1}).$$

From this it follows that

$$J(\pi) = \underbrace{\mathbb{E}_\pi\left[\sum_{n=0}^{N-1} R_{n+1} \,\middle|\, X_0 = x_0\right]}_{\text{Look familiar?!}} = v_\pi^1(x_0).$$

Market Making: Heuristic Strategies

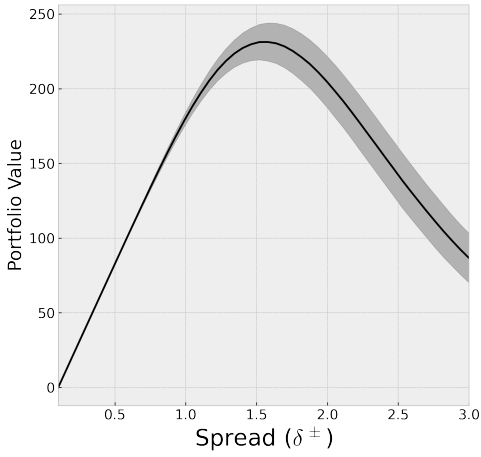
What if we always quote a fixed spread?

$$\delta_n^{\pm} \doteq \delta^{\pm}, \quad \forall n \geqslant 0.$$

Market Making: Heuristic Strategies

What if we always quote a fixed spread?

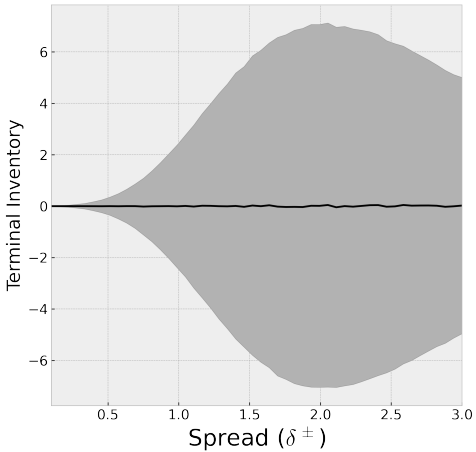
$$\delta_n^{\pm} \doteq \delta^{\pm}, \quad \forall n \geqslant 0.$$



Market Making: Heuristic Strategies

What if we always quote a fixed spread?

$$\delta_n^{\pm} \doteq \delta^{\pm}, \quad \forall n \geqslant 0.$$



Market Making: Heuristic Strategies

Can we avoid large positions?

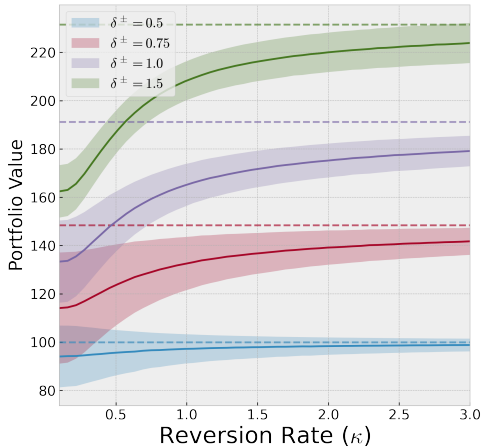
$$\delta_n^{\pm} \doteq \delta^{\pm} \cdot \exp\left(-\frac{1}{\kappa}\left[\Omega_n\right]_{\pm}\right).$$

Market Making: Heuristic Strategies

Can we avoid large positions?

$$\delta_n^{\pm} \doteq \delta^{\pm} \cdot \exp\left(-\frac{1}{\kappa}\left[\Omega_n\right]_{\pm}\right).$$

$\Longrightarrow$ *Induce mean reversion in* Ω_n.

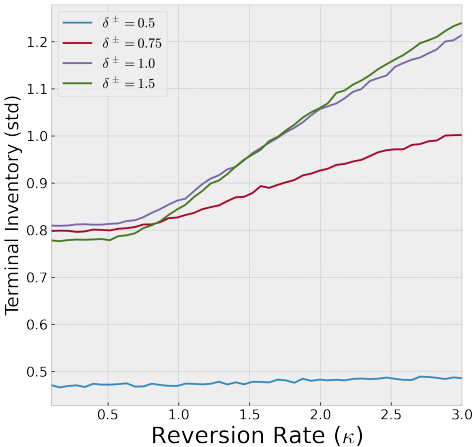


Market Making: Heuristic Strategies

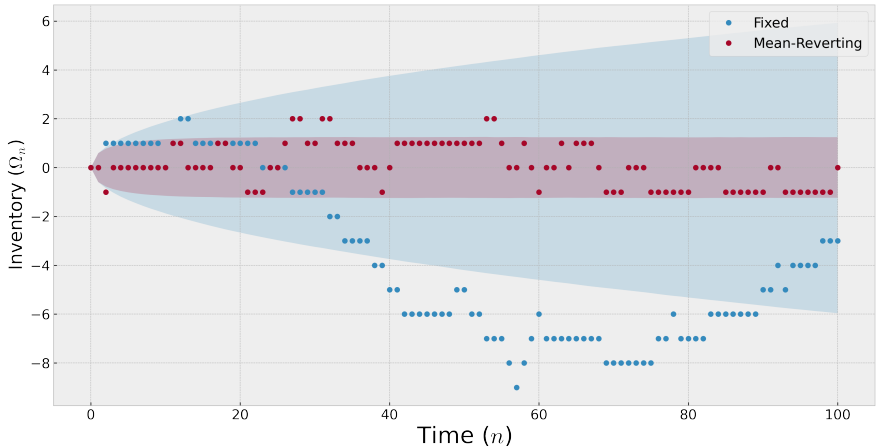
Can we avoid large positions?

$$\delta_n^{\pm} \doteq \delta^{\pm} \cdot \exp\left(-\frac{1}{\kappa}\left[\Omega_n\right]_{\pm}\right).$$

$\Longrightarrow$ *Induce mean reversion in* Ω_n.



Market Making: Heuristic Strategies



Market Making: Learnt Strategies

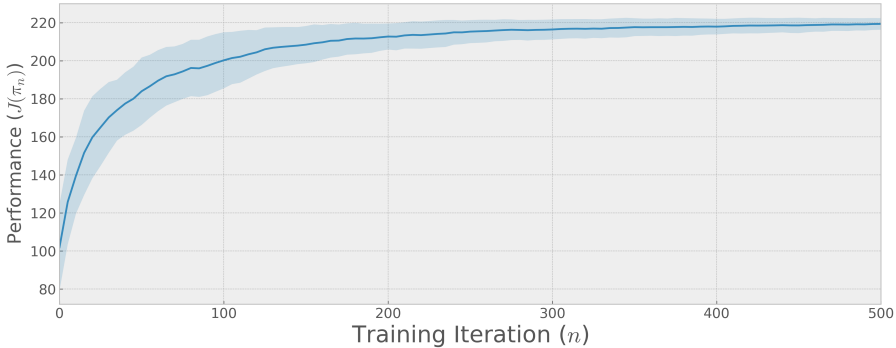
Alternatively, we can learn a policy![17]

- Represent π_θ using a cubic polynomial.
  - Yes, this means we only need 8 parameters!
- Use the REINFORCE algorithm:
  - Generate 200 sample paths.
  - Take the average gradient for better stability.

[17] *Happy to provide the code :)*

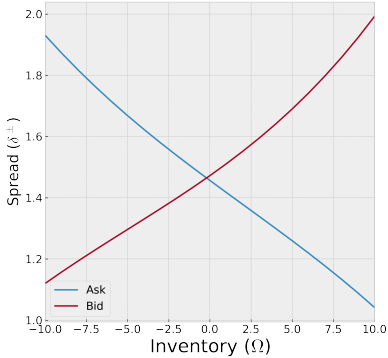
Market Making: Learnt Strategies

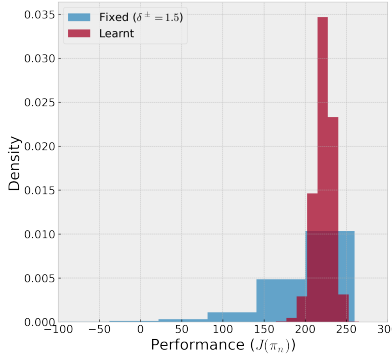
The performance improves smoothly over time (std over random seeds)!



Market Making: Learnt Strategies

The policy is smooth and interpretable, and also outperforms a fixed benchmark!





Gradient Factorisation

Scalability Issues

Policy gradient methods do not scale.

- We have to leverage prior knowledge!

Action-Dimensionality
If the dimensionality of U is large, meaningful exploration is hard.

Objective-Multiplicity
If $J(\pi)$ is multi-objective, you get high variance and chattering.

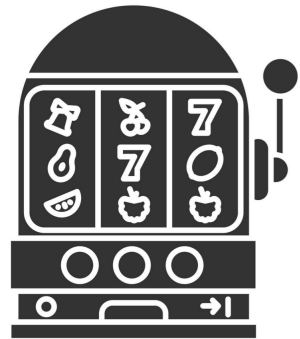
Scalability Issues

Let's say you're faced with n slot machines...

- you select $m \leqslant n$ arms...
- and see a **scalar reward** of $R = 5$.

Which arms were responsible?

- What if all arms gave positive reward?
- What if only one arm gave positive reward?



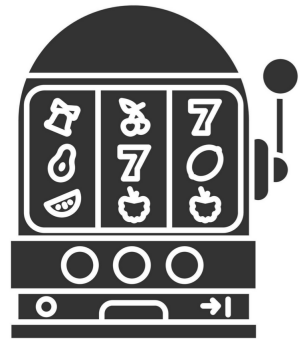
Scalability Issues

In reality, we see a vector, say:

$$\boldsymbol{R} = [0, \ldots, 0, 1, 0, \ldots 0] \, .$$

The standard MDP formulation ignores this!

- So how do we leverage this structure?



Scalability Issues

Market Making

Cross-Currency Arbitrage

Multi-Venue Optimal Execution

Water Reservoir Control

Elevator/Job Scheduling

Energy Consumption Optimisation

Influence Networks

Consider a *scalarised multi-objective MDP* where

$$J(\pi) = \mathbb{E}_\pi \left[q_\pi(X_0, U_0) \doteq \langle \boldsymbol{\lambda}, \boldsymbol{q}_\pi(X_0, U_0) \rangle = \sum_{i=1}^{m} \lambda_i q_{\pi,i}(X_0, U_0) \,\middle|\, X_0 \sim d_0 \right],$$

$$= \sum_{i=1}^{m} \lambda_i \mathbb{E}_\pi [q_{\pi,i}(X_0, U_0) \mid X_0 \sim d_0].$$

Influence Networks

Consider a *scalarised multi-objective MDP* where

$$J(\pi) = \mathbb{E}_\pi \left[q_\pi(X_0, U_0) \doteq \langle \boldsymbol{\lambda}, \boldsymbol{q}_\pi(X_0, U_0) \rangle = \sum_{i=1}^{m} \lambda_i q_{\pi,i}(X_0, U_0) \,\middle|\, X_0 \sim d_0 \right],$$
$$= \sum_{i=1}^{m} \lambda_i \mathbb{E}_\pi [q_{\pi,i}(X_0, U_0) \mid X_0 \sim d_0].$$

The value function breaks down into m distinct terms:

- This will be familiar to those who've studied *value decomposition networks*.

Influence Networks

Consider a *scalarised multi-objective MDP* where

$$J(\pi) = \mathbb{E}_\pi \left[q_\pi(X_0, U_0) \doteq \langle \boldsymbol{\lambda}, \boldsymbol{q}_\pi(X_0, U_0) \rangle = \sum_{i=1}^{m} \lambda_i q_{\pi,i}(X_0, \sigma_i(U_0)) \,\middle|\, X_0 \sim d_0 \right],$$
$$= \sum_{i=1}^{m} \lambda_i \mathbb{E}_\pi [q_{\pi,i}(X_0, \sigma_i(U_0)) \mid X_0 \sim d_0] \,.$$

What if each term $q_{\pi,i}$ only depends on a subset of the action U?

Influence Networks

This suggests a *linear-algebraic reformulation*:

$$J(\pi) = \mathbb{E}_\pi[q_\pi(X_0, U_0) \doteq \boldsymbol{K}\boldsymbol{\lambda} \circ \boldsymbol{q}_\pi(X_0, U_0) \,|\, X_0 \sim d_0]\,,$$

where

$$\boldsymbol{K} \in \mathbb{B}^{m \times m} \qquad \text{and} \qquad \begin{aligned} \boldsymbol{\lambda} &\doteq [\lambda_1, \ldots, \lambda_m] \in \mathbb{R}^m, \\ \boldsymbol{q}_\pi(x, u) &\doteq [q_{\pi,1}(x, u), \ldots, q_{\pi,m}(x, u)] \in \mathbb{R}^m. \end{aligned}$$

Influence Networks

This suggests a *linear-algebraic reformulation*:

$$J(\pi) = \mathbb{E}_\pi[q_\pi(X_0, U_0) \doteq \boldsymbol{K}\boldsymbol{\lambda} \circ \boldsymbol{q}_\pi(X_0, U_0) \,|\, X_0 \sim d_0]\,,$$

where

$$\underbrace{\boldsymbol{K} \in \mathbb{B}^{m \times m}}_{\text{Influence Matrix}} \qquad \text{and} \qquad \boldsymbol{\lambda} \doteq [\lambda_1, \ldots, \lambda_m] \in \mathbb{R}^m,$$

$$\boldsymbol{q}_\pi(x, u) \doteq [q_{\pi,1}(x, u), \ldots, q_{\pi,m}(x, u)] \in \mathbb{R}^m.$$

Influence Networks

Example

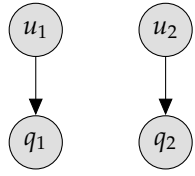
Consider a two-dimensional bandit of the form:

$$q(\cdot, \boldsymbol{u}) \doteq q_1(\cdot, u_1) + q_2(\cdot, u_2) = -|u_1 - c_1| - |u_2 - c_2|\,.$$

Influence Matrix

$$\boldsymbol{K} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Influence Network



Influence Networks

Example

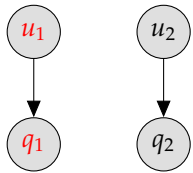
Consider a two-dimensional bandit of the form:

$$q(\cdot, \boldsymbol{u}) \doteq q_1(\cdot, u_1) + q_2(\cdot, u_2) = -|u_1 - c_1| - |u_2 - c_2|\,.$$

Influence Matrix

$$\boldsymbol{K} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Influence Network



Influence Networks

Example

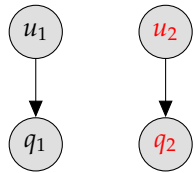
Consider a two-dimensional bandit of the form:

$$q(\cdot, \boldsymbol{u}) \doteq q_1(\cdot, u_1) + q_2(\cdot, u_2) = -|u_1 - c_1| - |u_2 - c_2|.$$

Influence Matrix

$$\boldsymbol{K} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Influence Network



Influence Networks

Example

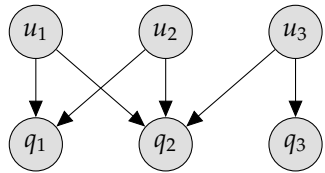
Consider an MDP with cross-dependency structure:

$$q(x, \boldsymbol{u}) \doteq \lambda_1 q_1(x, (u_1, u_2)) + \lambda_2 q_2(x, \boldsymbol{u}) + \lambda_3 q_3(x, u_3).$$

Influence Matrix

$$\boldsymbol{K} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Influence Network



Influence Networks

Example

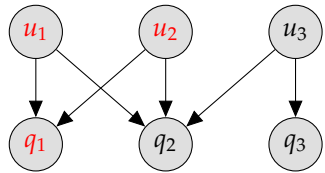
Consider an MDP with cross-dependency structure:

$$q(x, \boldsymbol{u}) \doteq \lambda_1 q_1(x, (u_1, u_2)) + \lambda_2 q_2(x, \boldsymbol{u}) + \lambda_3 q_3(x, u_3).$$

Influence Matrix

$$\boldsymbol{K} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Influence Network



Influence Networks

Example

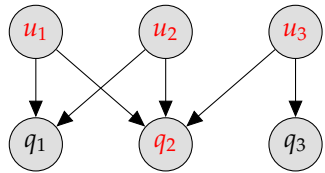
Consider an MDP with cross-dependency structure:

$$q(x, \boldsymbol{u}) \doteq \lambda_1 q_1(x, (u_1, u_2)) + \lambda_2 q_2(x, \boldsymbol{u}) + \lambda_3 q_3(x, u_3).$$

Influence Matrix

$$\boldsymbol{K} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Influence Network



Influence Networks

Example

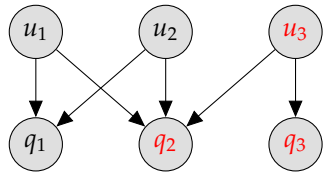
Consider an MDP with cross-dependency structure:

$$q(x, \boldsymbol{u}) \doteq \lambda_1 q_1(x, (u_1, u_2)) + \lambda_2 q_2(x, \boldsymbol{u}) + \lambda_3 q_3(x, u_3).$$

Influence Matrix

$$\boldsymbol{K} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Influence Network



Policy Factorisation

So how do we embed this knowledge in our algorithms?

[18] *For an MDP, we can always do this without affecting the optimum, though we may lose some generality…*

Policy Factorisation

So how do we embed this knowledge in our algorithms?

- We don't train actions, we train policies…

[18] *For an MDP, we can always do this without affecting the optimum, though we may lose some generality…*

Policy Factorisation

So how do we embed this knowledge in our algorithms?

- We don't train actions, we train policies…
- So we must first *impose the independence in our policy distribution*.

[18] *For an MDP, we can always do this without affecting the optimum, though we may lose some generality…*

Policy Factorisation

So how do we embed this knowledge in our algorithms?

- We don't train actions, we train policies…
- So we must first *impose the independence in our policy distribution*.

Using the σ/subset notation, we express the policy as a product distribution:[18]

$$\pi(\boldsymbol{u} \mid x; \theta) = \prod_{i=1}^{N} \pi_i(\sigma_i^{\pi}(\boldsymbol{u}) \mid x; \theta)\,.$$

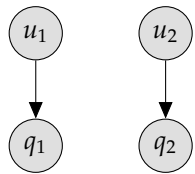
[18] *For an MDP, we can always do this without affecting the optimum, though we may lose some generality…*

Policy Factorisation

Recall the two-dimensional bandit:

$$q(\cdot, \boldsymbol{u}) = q_1(\cdot, u_1) + q_2(\cdot, u_2).$$

Influence Network



Policy Factorisation

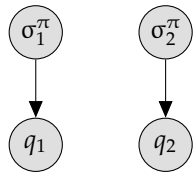
Recall the two-dimensional bandit:

$$q(\cdot, \boldsymbol{u}) = q_1(\cdot, u_1) + q_2(\cdot, u_2).$$

We can naturally separate into:

$$\sigma_1^\pi(\boldsymbol{u}) = (u_1),$$
$$\sigma_2^\pi(\boldsymbol{u}) = (u_2).$$

***Factored* Influence Network**

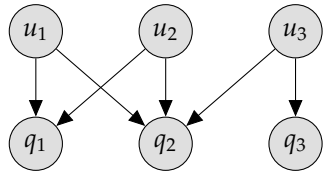


Policy Factorisation

Recall the mixed case:

$$\lambda_1 q_1(x, (u_1, u_2)) + \lambda_2 q_2(x, \boldsymbol{u}) + \lambda_3 q_3(x, u_3).$$

Influence Network



Policy Factorisation

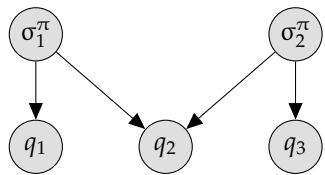
Recall the mixed case:

$$\lambda_1 q_1(x, (u_1, u_2)) + \lambda_2 q_2(x, \boldsymbol{u}) + \lambda_3 q_3(x, u_3).$$

We can naturally separate into:

$$\sigma_1^\pi(\boldsymbol{u}) = (u_1, u_2),$$
$$\sigma_2^\pi(\boldsymbol{u}) = (u_3).$$

***Factored* Influence Network**



Gradient Factorisation

Recall **Sutton's vanilla policy gradient**:[19]

$$\begin{aligned}\nabla_\theta J(\pi) &= \mathbb{E}_\pi[\nabla_\theta \log \pi(\boldsymbol{U} \mid X)\, q(X, \boldsymbol{U})]\,, \\ &\doteq \mathbb{E}_\pi[\boldsymbol{S}(X, \boldsymbol{U})\, q(X, \boldsymbol{U})]\,.\end{aligned}$$

[19] *Yes, I've abused notation, but it makes things cleaner… bear with me…*

Gradient Factorisation

Recall **Sutton's vanilla policy gradient**:[19]

$$\begin{aligned}\nabla_\theta J(\pi) &= \mathbb{E}_\pi[\nabla_\theta \log \pi(\boldsymbol{U} \mid X)\, q(X, \boldsymbol{U})]\,, \\ &\doteq \mathbb{E}_\pi[\boldsymbol{S}(X, \boldsymbol{U})\, q(X, \boldsymbol{U})]\,.\end{aligned}$$

In this case we have the following properties:

- The target $q(X, \boldsymbol{U})$ is a scalar.
- The scores $\boldsymbol{S}(X, \boldsymbol{U})$ are an $|\theta| \times 1$ real vector.

[19] *Yes, I've abused notation, but it makes things cleaner…bear with me…*

Gradient Factorisation

The **factored policy gradient** naturally extends this:

$$\nabla_\theta J(\pi) = \mathbb{E}_\pi[\boldsymbol{S}(X, \boldsymbol{U})\, \boldsymbol{K}\, \boldsymbol{\lambda} \circ \boldsymbol{q}(X, \boldsymbol{U})]\,.$$

Gradient Factorisation

The **factored policy gradient** naturally extends this:

$$\nabla_\theta J(\pi) = \mathbb{E}_\pi[\boldsymbol{S}(X, \boldsymbol{U})\, \boldsymbol{K}\, \boldsymbol{\lambda} \circ \boldsymbol{q}(X, \boldsymbol{U})]\,.$$

In this case we have the following properties:

- The target $\boldsymbol{\lambda} \circ \boldsymbol{q}(X, \boldsymbol{U})$ is an $m \times 1$ real vector.
- The scores $\boldsymbol{S}(X, \boldsymbol{U})$ are an $|\theta| \times n$ real matrix.
- The influence $\boldsymbol{K}$ is an $n \times m$ real matrix.

Gradient Factorisation

The **factored policy gradient** naturally extends this:

$$\nabla_{\boldsymbol{\theta}} J(\pi) = \mathbb{E}_{\pi}[\boldsymbol{S}(X, \boldsymbol{U})\, \boldsymbol{K} \circ \boldsymbol{\lambda}\, \boldsymbol{q}(X, \boldsymbol{U})]\,.$$

These are *equivalent in expectation due to the baseline trick,* with:

$$b_i(x, \boldsymbol{u}) \doteq [(\mathbf{1} - \boldsymbol{K})\, \boldsymbol{\lambda} \circ \boldsymbol{q}(x, \boldsymbol{u})]_i\,.$$

Market Making

Recall the mark-to-market reward function:

$$R_{n+1} = \underbrace{\delta_n^- \Delta\Omega_n^-}_{\text{Ask}} + \underbrace{\delta_n^+ \Delta\Omega_n^+}_{\text{Bid}} + \underbrace{\Omega_{n+1}\Delta Z_n(1 - \beta\Omega_{n+1})}_{?}.$$

Market Making

Recall the mark-to-market reward function:

$$R_{n+1} = \underbrace{\delta_n^- \Delta\Omega_n^-}_{\text{Ask}} + \underbrace{\delta_n^+ \Delta\Omega_n^+}_{\text{Bid}} + \underbrace{\Omega_{n+1}\Delta Z_n(1 - \beta\Omega_{n+1})}_{?}.$$

We can of course unroll ? to further separate variables:

$$R_n = R_n^- + R_n^+ + R_n^0,$$

where

$$R_{n+1}^{\pm} = \Delta Q_n^{\pm} \cdot (\delta_n^{\pm} \pm \Delta Z_n) + \beta \cdot \Delta Q_n^{\pm} \cdot (\Delta Z_n)^2 \cdot (2Q_n \pm \Delta Q_n^{\pm}),$$
$$R_{n+1}^0 = Q_n \cdot \Delta Z_n - \beta \cdot (\Delta Z_n)^2 \cdot (Q_n^2 - 2 \cdot \Delta Q_n^+ \cdot \Delta Q_n^-).$$

Market Making

We now have a clear split:

$$R_n = R_n^- + R_n^+ + R_n^0,$$

where

$$\begin{aligned}\operatorname{corr}\left(R_{n+1}^+, R_{n+1}^- \middle| X_n, U_n\right) &= 0,\\ \operatorname{corr}\left(R_{n+1}^\pm, R_{n+1}^0 \middle| X_n, U_n\right) &\neq 0.\end{aligned}$$

Market Making

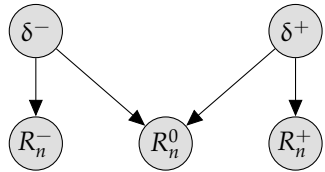
We now have a clear split:

$$R_n = R_n^- + R_n^+ + R_n^0,$$

where

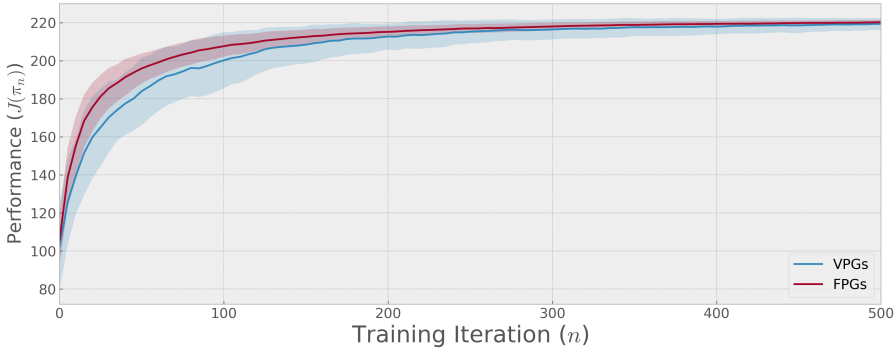
$$\operatorname{corr}\left(R_{n+1}^+, R_{n+1}^- \middle| X_n, U_n\right) = 0,$$
$$\operatorname{corr}\left(R_{n+1}^\pm, R_{n+1}^0 \middle| X_n, U_n\right) \neq 0.$$

The resulting *influence network*:



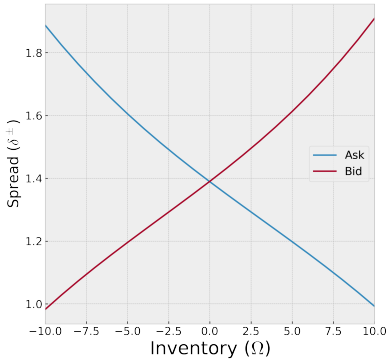
Market Making

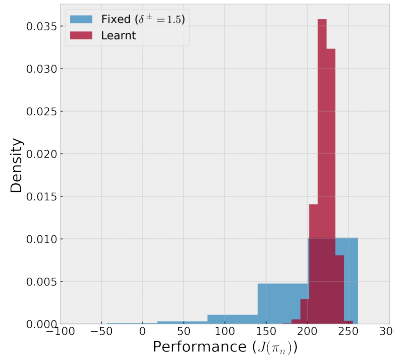
Factoring gradients reduces variance!



Market Making

The (finite-time) policy is also more symmetric due to reduced variance!





Transition Acyclicity

How hard is it to solve MDPs in general?

Complexity

- Littman et al. (2013)

394

On the Complexity of Solving Markov Decision Problems

Michael L. Littman, Thomas L. Dean, Leslie Pack Kaelbling
Department of Computer Science
Brown University
115 Waterman Street
Providence, RI 02912, USA
Email: {mlittman,tld,lpk}@cs.brown.edu

Abstract

Markov decision problems (MDPs) provide the foundations for a number of problems of interest to AI researchers studying automated planning and reinforcement learning. In this paper, we summarize results regarding the complexity of solving MDPs and the running time of MDP solution algorithms. We argue that, although MDPs can be solved efficiently in theory, more study is needed to reveal practical algorithms for solving large problems quickly. To encourage future research, we sketch some alternative methods of analysis that rely on the structure of MDPs.

1 INTRODUCTION

A *Markov decision process* is a controlled stochastic process satisfying the Markov property with costs assigned to state transitions. A *Markov decision problem* is a Markov decision process together with a performance criterion. A solution to a Markov decision problem is a *policy*, mapping states to actions, that (perhaps stochastically) determines state transitions to minimize the cost according to the performance criterion. Markov decision problems (MDPs) provide the theoretical foundations for decision-theoretic planning, reinforcement learning, and other sequential decision-making tasks of interest to researchers and practitioners in artificial intelligence and operations research (Dean et al., 1995b). MDPs employ dynamical models based on well-understood stochastic processes and performance criteria based on established theory in operations research, economics, combinatorial optimization, and the social sciences (Puterman, 1994).

It would seem that MDPs exhibit special structure that might be exploited to expedite their solution. In investment planning, for example, often the initial state is known with certainty (the current price for a stock or commodity) and as a result the set of likely reachable states (future prices) and viable investment strategies in the near-term future is considerably restricted. In general, notions of time, action, and reachability in state space are inherent characteristics of MDPs that might be exploited to produce efficient algorithms for solving them. It is important that we understand the computational issues involved in these sources of structure to get some idea of the prospects for efficient sequential and parallel algorithms for computing both exact and approximate solutions.

This paper summarizes some of what is known (and unknown but worth knowing) about the computational complexity of solving MDPs. Briefly, any MDP can be represented as a linear program (LP) and solved in polynomial time. However, the order of the polynomials is large enough that the theoretically efficient algorithms are not efficient in practice. Of the algorithms specific to solving MDPs, none is known to run in worst-case polynomial time. However, algorithms and analyses to date have made little use of MDP-specific structure, and results in related areas of Monte Carlo estimation and Markov chain theory suggest promising avenues for future research. We begin by describing the basic class of problems.

2 MARKOV DECISION PROBLEMS

For our purposes, a *Markov decision process* is a four-tuple $(\Omega_S, \Omega_A, p, c)$, where Ω_S is the *state space*, Ω_A is the *action space*, p is the state-transition probability-distribution function, and c is the instantaneous-cost function.

The state-transition function is defined as follows: for all $i, j \in \Omega_S, k \in \Omega_A$,

$$p_{ij}^k = \Pr\{S_t = j | S_{t-1} = i, A_t = k\}$$

where S_t (A_t) is a random variable denoting the state (action) at time t. The cost c_i^k is defined to be the cost of taking action k from state i.

Let $N = |\Omega_S|$ and $M = |\Omega_A|$. For some of the complexity results, it will be necessary to assume that p and c are encoded using $N \times N \times M$ tables of rational numbers. We let B be the maximum number of bits required to represent any component of p or c. In

Complexity

- Littman et al. (2013)
- Fearnley and Savani (2014)

The Complexity of the Simplex Method

John Fearnley and Rahul Savani
University of Liverpool

Abstract. The simplex method is a well-studied and widely-used pivoting method for solving linear programs. When Dantzig originally formulated the simplex method, he gave a natural pivot rule that pivots into the basis a variable with the most violated reduced cost. In their seminal work, Klee and Minty showed that this pivot rule takes exponential time in the worst case. We prove two main results on the simplex method. Firstly, we show that it is PSPACE-complete to find the solution that is computed by the simplex method using Dantzig's pivot rule. Secondly, we prove that deciding whether Dantzig's rule ever chooses a specific variable to enter the basis is PSPACE-complete. We use the known connection between Markov decision processes (MDPs) and linear programming, and an equivalence between Dantzig's pivot rule and a natural variant of policy iteration for average-reward MDPs. We construct MDPs and show PSPACE-completeness results for single-switch policy iteration, which in turn imply our main results for the simplex method.

1 Introduction

Linear programming is a fundamental technique in computer science, and the simplex method is a widely used technique for solving linear programs. The simplex method requires a *pivot rule* that determines which variable is pivoted into the basis in each step. Dantzig's original formulation of the simplex method used a particularly natural pivot rule: in each step, the non-basic variable with the most negative reduced cost is chosen to enter the basis [5]. We will call this *Dantzig's pivot rule*. Klee and Minty have shown that Dantzig's pivot rule takes exponential time in the worst case [17].

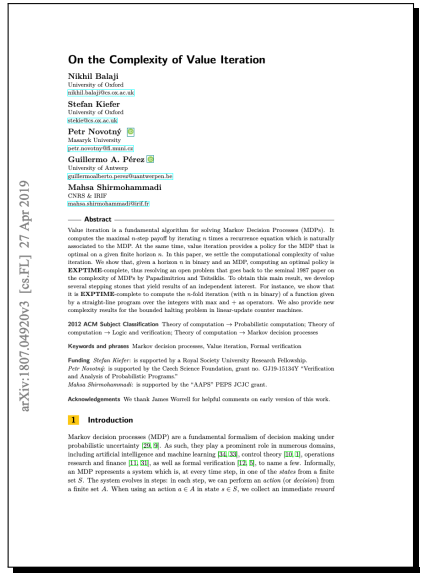
The simplex method is a member of a of much wider class of *local search* algorithms. The complexity class PLS, which was introduced by Johnson, Papadimitriou, and Yannakakis, captures problems where a locally optimal solution can be found by a local search algorithm [16]. PLS has matured into a robust class, and there is now a wide range of problems that are known to be PLS-complete [15, 19]. It is widely believed that PLS-complete problems do not admit polynomial-time algorithms.

To show that a problem lies in PLS, we must provide three polynomial-time functions: a function A that produces a candidate solution, a function B that assigns an value to each candidate solution, and a function C that for each candidate solution, either produces a neighbouring candidate solution with higher value, or reports that no such candidate solution exists. Thus, each PLS problem comes equipped with a *natural algorithm* that executes A to find an initial candidate solution, and then iterates C until a local optimum is found. For some problems in PLS, it is known that it is PSPACE-complete to find the solution that is computed by the natural algorithm [22, 23]. So far, this is only known to hold for problems that are *tight PLS-complete* (see, e.g., [28]), which is a stronger form of PLS-completeness, or for problems that are suspected to be tight PLS-complete, such as the local max-cut problem on graphs of degree four [20].

Obviously, since linear programming is in P, it cannot be PLS-complete unless PLS=P. Despite this fact, in the first main theorem of this paper, we show that it is PSPACE-complete to compute the solution found by the simplex method equipped with Dantzig's pivot rule. Given a linear program $\mathcal{L}$, an initial basic feasible solution b, and a variable v, the problem DantzigLpSol$(\mathcal{L}, b, v)$ asks the following question: if Dantzig's pivot rule is started at basis b, and finds an optimal solution

Complexity

- Littman et al. (2013)
- Fearnley and Savani (2014)
- Balaji et al. (2018)



On the Complexity of Value Iteration

Nikhil Balaji
University of Oxford
nikhil.balaji@cs.ox.ac.uk

Stefan Kiefer
University of Oxford
sk@cs.ox.ac.uk

Petr Novotný
Masaryk University
petr.novotny@fi.muni.cz

Guillermo A. Pérez
University of Antwerp
guillermoalberto.perez@uantwerpen.be

Mahsa Shirmohammadi
CNRS & IRIF
mahsa.shirmohammadi@irif.fr

Abstract

Value iteration is a fundamental algorithm for solving Markov Decision Processes (MDPs). It computes the maximal n-step payoff by iterating n times a recurrence equation which is naturally associated to the MDP. At the same time, value iteration provides a policy for the MDP that is optimal on a given finite horizon n. In this paper, we settle the computational complexity of value iteration. We show that, given a horizon n in binary and an MDP, computing an optimal policy is **EXPTIME**-complete, thus resolving an open problem that goes back to the seminal 1987 paper on the complexity of MDPs by Papadimitriou and Tsitsiklis. To obtain this main result, we develop several stepping stones that yield results of an independent interest. For instance, we show that it is **EXPTIME**-complete to compute the n-fold iteration (with n in binary) of a function given by a straight-line program over the integers with max and $+$ as operators. We also provide new complexity results for the bounded halting problem in linear-update counter machines.

2012 *ACM* Subject Classification Theory of computation → Probabilistic computation; Theory of computation → Logic and verification; Theory of computation → Markov decision processes

Keywords and phrases Markov decision processes, Value iteration, Formal verification

Funding *Stefan Kiefer*: is supported by a Royal Society University Research Fellowship.
Petr Novotný: is supported by the Czech Science Foundation, grant no. GJ19-15134Y "Verification and Analysis of Probabilistic Programs"
Mahsa Shirmohammadi: is supported by the "AAPS" PEPS JCJC grant.

Acknowledgements We thank James Worrell for helpful comments on early version of this work.

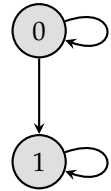
1 Introduction

Markov decision processes (MDP) are a fundamental formalism of decision making under probabilistic uncertainty [29, 9]. As such, they play a prominent role in numerous domains, including artificial intelligence and machine learning [34, 33], control theory [10, 1], operations research and finance [11, 31], as well as formal verification [12, 4], to name a few. Informally, an MDP represents a system which is, at every time step, in one of the *states* from a finite set S. The system evolves in steps: in each step, we can perform an *action* (or *decision*) from a finite set A. When using an action $a \in A$ in state $s \in S$, we collect an immediate *reward*

Complexity: Cycles

Consider the following undiscounted MRP:

$$\boldsymbol{P} \doteq \begin{bmatrix} \nu & 1-\nu \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{R} \doteq \begin{bmatrix} \rho & 0 \\ 0 & 0 \end{bmatrix}.$$



We can of course solve this analytically…

$$v(0) = \underbrace{\nu\,[\rho + v(0)]}_{\text{Cycle}} + \underbrace{(1-\nu)\,[0 + v(1)]}_{\text{Termination}},$$

such that

$$v(0) = \frac{\nu\rho}{1-\nu}.$$

Complexity: Cycles

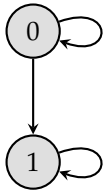
But how does value iteration perform?

Starting with $v_0(0) = 0$, we get

$$v_1(0) = \nu\rho, \qquad v_2(0) = \nu\rho + \nu^2\rho,$$

and eventually

$$v_k(0) = \sum_{i=1}^{k} \nu^i \rho.$$



Complexity: Cycles

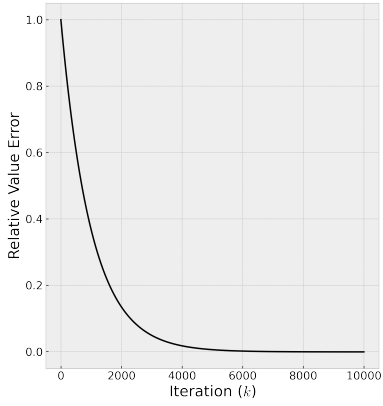
But how does value iteration perform?

Of course, we end up in the same place…

$$\lim_{k\to\infty} v_k(0) = \frac{\nu\rho}{1-\nu}.$$

But it will take a long time to get there!

- Cycles are bad!



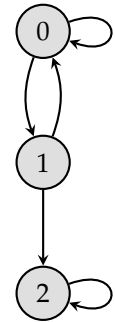
Complexity: Bootstrapping

But there's a bigger issue!

- *VI relies heavily on bootstrapping*.

When estimating $v(\cdot)$ for multiple states:

- Cycles get longer.
- Feedback is slow to propagate.
- And intermediate estimates are erroneous.



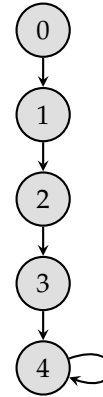
Acyclicity

What if there weren't any cycles?

- Could we do better?

The key is hidden in the VI algorithm:

- *We are free to choose the order of updates!*



Acyclicity

Theorem (Optimal Backup Order[20])

If an MDP is acyclic, then there exists an optimal backup order. By applying the optimal order, the optimal value function can be found with each state needing only one backup.

[20] *Dimitri Bertsekas. Dynamic Programming and Optimal Control. Vol. 2. Athena Scientific, 2012.*

Acyclicity

Theorem (Optimal Backup Order[21])

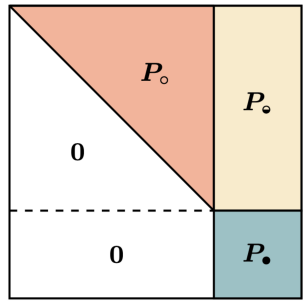
If an MDP is acyclic, then there exists an optimal backup order. By applying the optimal order, the optimal value function can be found with each state needing only one backup.

[21] *Dimitri Bertsekas. Dynamic Programming and Optimal Control. Vol. 2. Athena Scientific, 2012.*

Acyclicity

Why is this true?

- Acyclicity implies $\boldsymbol{P}$ is upper-triangular!



And we can take advantage of this:

- Perform VI updates according to the topological order.
- "Start from the end, and work backwards."

Recall the backwards induction algorithm!

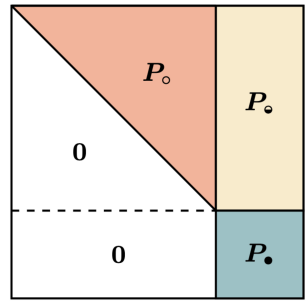
Acyclicity

It turns out, many problems are like this!

1. Routing problems.
2. Finite resource problems.
3. *Optimal liquidation.*

In other cases, we can impose it.

- This leads to sub-optimality.
- But drastically reduces training time.



Acyclicity: Optimal Liquidation

Let's consider a concrete example.

- Discrete optimal liquidation problem.
- *Where the agent has to trade every step.*

Acyclicity: Optimal Liquidation

Let's consider a concrete example.

- Discrete optimal liquidation problem.
- *Where the agent has to trade every step.*

Inventory Process

- Agent chooses $0 < U_n \leqslant \Omega_n$.
- Inventory transitions:

$$\Omega_n \mapsto \Omega_n - U_n.$$

Acyclicity: Optimal Liquidation

Let's consider a concrete example.

- Discrete optimal liquidation problem.
- *Where the agent has to trade every step.*

Inventory Process

- Agent chooses $0 < U_n \leqslant \Omega_n$.
- Inventory transitions:

$$\Omega_n \mapsto \Omega_n - U_n.$$

Price Process

- Trinomial random walk: $\boldsymbol{p} \in \Delta^2$.
- With imposed reflection.

Acyclicity: Optimal Liquidation

Let's consider a concrete example.

- Discrete optimal liquidation problem.
- *Where the agent trades every step.*

Reward Process

$$R_n \doteq \underbrace{w_0 \cdot U_n \cdot (Z_n - Z_0)}_{\text{Excess PnL}} - \underbrace{w_1 \cdot U_n^2 - w_2 \cdot \Omega_n^2}_{\text{Penalty}}$$

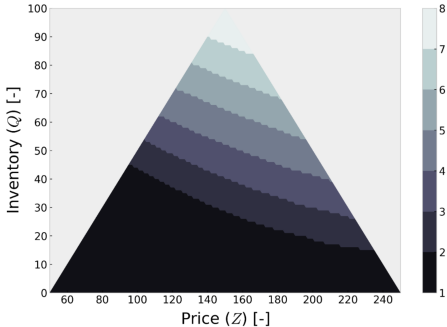
Acyclicity: Optimal Liquidation

Let us set the dynamics to

- Start with $\Omega_0 \doteq 100$ and $Z_0 \doteq 150$.
- Transitions with $\boldsymbol{p} \doteq [0.4, 0.2, 0.4]$.
- And reward weights $\boldsymbol{w} \doteq [1, 0.2, 0.002]$.

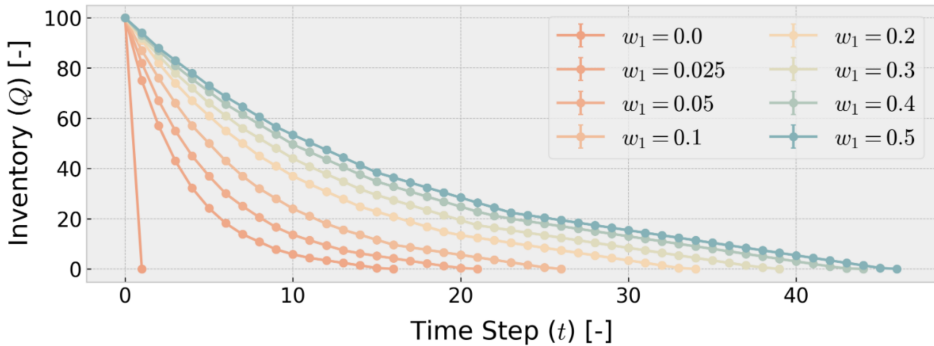
Optimal policy is pretty intuitive!

- Sell fast when prices are high.
- Decrease rate as inventory shrinks.



Reductivity

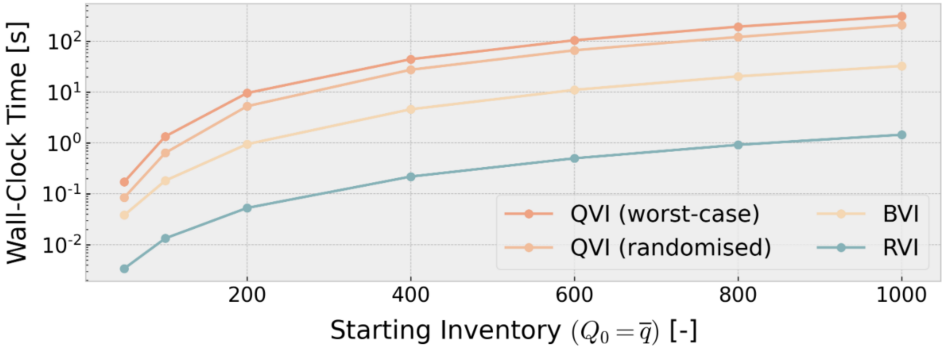
Increasing the penalty term w_1 slows the trading rate.



Reductivity

Accounting for acyclicity (RVI) gives OoM reductions in time!

- Time complexity of RVI forms a lower-bound.[22]



[22] *Leveraging structure for fun and profit…and saving time!*

Reductivity

So what's the theory behind it?

Reductivity

Recall: a set $B \in \mathfrak{B}(\mathsf{X})$ is reachable from $x \in \mathsf{X}$ if $P_n(x, B)$ for some $n > 0$.

We thus define the *n-step reachable state set* from $x \in \mathsf{X}$ as

$$\mathsf{F}_0(x) \doteq \{x\}\,,$$
$$\mathsf{F}_n(x) \doteq \bigcap \{B \in \mathfrak{B}(\mathsf{X}) \,:\, P_n(x, B) > 0\}\,.$$

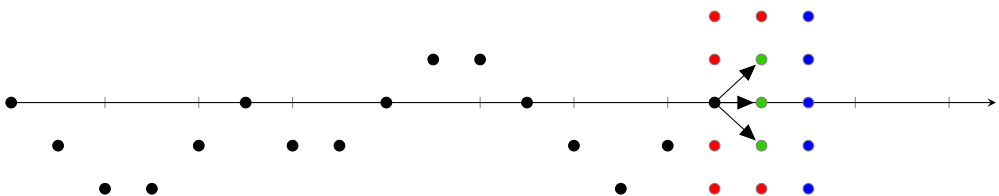
The *reachable state set* is the union thereof

$$\mathsf{F}(x) \doteq \bigcup_{n=0}^{\infty} \mathsf{F}_n(x) \quad \Longrightarrow \quad \mathsf{F}(B) \doteq \bigcup_{x \in B} \mathsf{F}(x)\,.$$

Reductivity

In the random walk, where $P(x, \{x+1\}) = P(x, \{x\}) = P(x, \{x-1\})$, we have

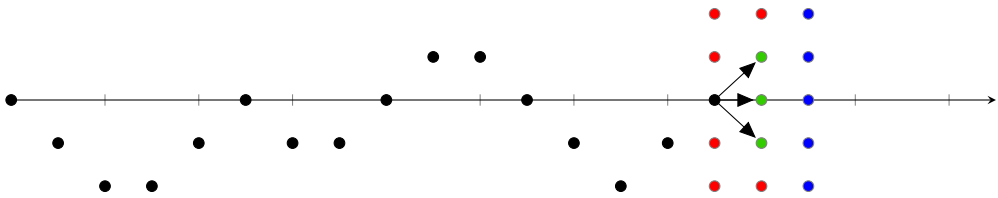
$$\mathsf{F}_1(x) = \{x-1, x, x+1\},$$
$$\mathsf{F}_2(x) = \{x-2, x-1, x, x+1, x+2\},$$
$$\vdots$$



Reductivity

In the random walk, where $P(x, \{x+1\}) = P(x, \{x\}) = P(x, \{x-1\})$, we have

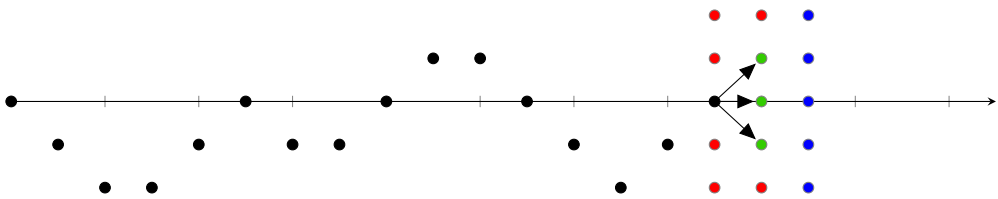
$$\mathsf{F}_n(x) = \{x-n, \ldots, x, \ldots, x+n\}.$$



Reductivity

In the random walk, where $P(x, \{x+1\}) = P(x, \{x\}) = P(x, \{x-1\})$, we have

$$\mathsf{F}(x) = \bigcup_{n=0}^{\infty} \{x-n, \ldots, x+n\} = \mathbb{Z}.$$



Reductivity

Is there a succinct way to describe reachable sets?

Reductivity

Is there a succinct way to describe reachable sets? Yes!

For a measure $\mu : \mathfrak{B}(\mathsf{X}) \to [0, \infty)$, the *reachability potential* is

$$\phi_\mu(x) \doteq \mu \circ \mathsf{F}(x)\,,$$

with "discrete derivative" given by

$$\psi_\mu(x, x') \doteq \phi_\mu(x') - \phi_\mu(x) = -\mu\big(\mathsf{F}(x) \setminus \mathsf{F}(x')\big) \leqslant 0.$$

Reductivity

If we now define the set of non-absorbing cyclic states,

$$\mathsf{L}(A) \doteq \{x \in A : 0 < \mathbb{P}(x, \{x\}) < 1\},$$

then we can now define an RMC.

Reductivity

If we now define the set of non-absorbing cyclic states,

$$\mathsf{L}(A) \doteq \{x \in A : 0 < \mathbb{P}(x, \{x\}) < 1\},$$

then we can now define an RMC.

Definition (RMC)

Let μ be a measure. Then, an MC is said to be μ-reductive if $\mathsf{X}_\bullet \neq \{\}$ and

$$\begin{cases} \psi_\mu(x, x') = 0 & \text{for } x' \in \mathsf{X}_\bullet \cup \mathsf{L}(\{x\}), \\ \psi_\mu(x, x') < 0 & \text{otherwise,} \end{cases}$$

holds for all states $x \in \mathsf{X}$ and their successors $x' \in \mathsf{F}_1(x)$.

Reductivity

This is a sufficient condition for acyclicity!

- But also extends the concept to continuous state-spaces!

Definition (RMC)

Let μ be a measure. Then, an MC is said to be μ-reductive if $\mathsf{X}_\bullet \neq \{\}$ and

$$\begin{cases} \psi_\mu(x, x') = 0 & \text{for } x' \in \mathsf{X}_\bullet \cup \mathsf{L}(\{x\}), \\ \psi_\mu(x, x') < 0 & \text{otherwise,} \end{cases}$$

holds for all states $x \in \mathsf{X}$ and their successors $x' \in \mathsf{F}_1(x)$.

Reductivity

We can also apply to MDPs (see below):[23]

- *The corresponding algorithm, RVI, solves RMDPs in polynomial time!*

Definition (RMDP)

An MDP is said to be μ-reductive if every policy induces a μ-RMC.

[23] *Thomas Spooner et al. "Reductive MDPs: A Perspective Beyond Temporal Horizons". In: arXiv preprint arXiv:2205.07338 (2022).*

That’s all folks!