UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

Approximate Guarantee for Approximate Dynamic Programming

W.T.P. Lardinois

M.Sc. Thesis
December 2019

Supervisor:
Prof. dr. R.J. Boucherie

Applied Mathematics
Stochastic Operations Research (SOR)

Abstract

A Markov decision process (MDP) is a common way to model stochastic decision problems. Finding the optimal policy for an MDP is a challenging task. Therefore, approximation methods are used to obtain decent policies. Approximate dynamic programming (ADP) is an approximation method to obtain policies for an MDP. No approximate guarantees for ADP related to MDP exist yet. This thesis searches for an approximate guarantee for ADP by using an optimal stopping problem. In Chen & Goldberg, 2018 [10], an approximation method and approximate guarantees were obtained for optimal stopping problems.

A Markov chain over the policy space of the MDP is created to obtain an optimal stopping problem, denoted by OS-MDP. The method described by Chen & Goldberg applied on OS-MDP yields error bounds for solution methods and approximation methods of MDP. The estimation relates the policy found after N iterations to the policy obtained after M iterations, where $N < M$. This estimation has an error bound of $\frac{1}{k+1}$, where k is a parameter that determines the complexity of the computations. Solution methods discussed are; policy iteration, value iteration and ADP.

A card-game, called The Game [36], is used as a running example. The Game is modelled as MDP, approximated using ADP, and an error bound of ADP is obtained by using OS-MDP. One small numerical test of OS-MDP is performed, where $N = 5$ and $M = 10$ yields an error bound of 0.25, hence no more than a 25% improvement can be achieved.

Contents

Chapter 1

Introduction

1.1 Motivation and framework

A Markov decision process, or MDP for short, is a widely used model for stochastic decision problems. There are various fields where MDPs are used, for example, healthcare, logistics, games and machine learning. Finding the optimal policy for an MDP is a challenging task, and therefore receives significant academic interest. MDP can be solved using the Bellman equation [24]. Various algorithms have been created that find the optimal policy for a given MDP under a set of assumptions. Some example algorithms are value iteration, policy iteration and modified policy iteration [24].

An issue is that many practical MDPs cannot be solved in a reasonable amount of time using these algorithms, because the MDP is too large and complex to solve [18, 21]. This issue creates the need for approximation methods that find a good policy for large MDPs. Approximate Dynamic Programming (ADP) is an approximation method for MDP, which we will focus on in this thesis.

One problem with ADP is that determining the quality of a solution is difficult [23]. This problem could lead to cases where, without knowing, a very mediocre policy is used. Hence a method to assess the quality of a policy is desired. There are several to estimate the quality of a policy[22], for example, comparing different policies found by approximation methods. One method to determine the quality of a policy is by finding an approximation guarantee, which relates the policy obtained by the approximation method to the optimal policy of the MDP. For ADP, no such approximate guarantee currently exist [23].

In this thesis, we explore an approach to find an approximate guarantee for ADP: To create an optimal stopping problem from a general MDP, denoted by OS-MDP. The idea is to create a stochastic process over the policy space. We then search for the stopping time that minimises the expected costs. Solution methods for solving an MDP can be used to define the stochastic process of OS-MDP. The following methods are used to illustrate this: value iteration, policy iteration and ADP.

Since OS-MDP is an optimal stopping problem, methods for optimal stopping problems can be related to an MDP. In [10], an approximation method for the optimal stopping problem was created that includes approximation guarantees. This method is introduced and proven

in simpler terms. Additionally, we apply the approximation method to the OS-MDP model to obtain bounds for different solution methods, namely policy iteration, value iteration and ADP.

1.2 Running example: The Game

A game will be used as a running example throughout the thesis. The chosen game is a finite horizon game with finite-sized policy space, finite costs and finite-sized state-space, namely The Game, designed by White Goblin Games [36]. The Game is a cooperative or single-player card game where the player(s) has to play cards numbered 2 till 99 on 4 piles in the centre, following a set of playing rules. We consider only the single-player version, which makes The Game a discrete-time stochastic decision problem with a short time horizon. Finding the optimal strategy for The Game is difficult, since the state-space is large, making it a suitable candidate for testing approximation methods.
Another game used to test approximation methods is Tetris [6]. Tetris is introduced and modelled as MDP. The reason The Game is picked over Tetris as the running example is that, even though Tetris ends with probability one [4], the horizon length is unknown beforehand. The Game is modelled as MDP. We explain how ADP can be applied, and we obtain numerical results for The Game. The different policies obtained by ADP are compared to find what ADP settings perform best. Additionally, we test the OS-MDP model for The Game to obtain bounds, which is done for value iteration, policy iteration and ADP. Finally, we compute a simple error bound using OS-MDP with ADP.

1.3 Organisation of master thesis

In Chapter 2, a general literature overview is given. This chapter is split into different topics: MDP, optimal stopping problems and ADP. Additionally, the contribution of the OS-MDP model is stated as well as a brief comparison to similar models.
Chapter 3 contains the definition of a Markov decision process, including several exact solution methods. Additionally, approximate dynamic programming is introduced. Furthermore, The Game and Tetris are introduced and modelled as MDP. Then a description is given on how The Game can be approximated using ADP.
In Chapter 4, optimal stopping problems are introduced and defined. Additionally, we introduce an approximation method for optimal stopping problems, taken from Y. Chen & D. Goldberg 2018 [10], and state the approximate guarantee results.
Chapter 5 introduces the OS-MDP model formally. Several solution methods of MDP are implemented into the OS-MDP, and additional results and bounds are given.
Chapter 6 contains all results related to The Game, including ADP approximations and OS-MDP results.
Finally, the thesis is summarised, discussed and concluded in Chapter 7.

Chapter 2

Literature Research

This chapter gives a literature overview of several relevant topics. First, a discussion about Markov Decision Processes (MDP), their solution methods and possible applications.Second, we discuss the field of Optimal Stopping (OS). This overview includes practical applications and possible solution methods. Third, a discussion about the field of Approximate Dynamic Programming (ADP). ADP can be applied on both MDP and OS, which means it will contain references to both fields. Several practical examples and algorithms of ADP will be given. In section 2.4 we state the contribution of the thesis.

2.1 Markov decision process

MDPs o□er a way to model stochastic decision problems. We give a general description of an MDP. At every time step, the process is in some state. In this state, an action is picked and consequently process randomly moves to a new state. Then some cost is obtained dependent on the action and/or transition from state to state. The goal is to minimise the costs, which is achieved by choosing the best action in every state. Combining all actions of all states combined yields a policy. Hence we want to find a policy that minimises our costs. In MDP it is assumed that given the present, then the future is independent of the past. This implies that the state only consists of the present and not the past.
An MDP can be both discrete or continuous time. In this thesis, we will only consider the discrete-time stochastic control problems. For a thorough treatment of MDP, we refer the reader to [24]. MDP can be used to model various problems, we state several recent examples. For an introduction on how MDP can be applied to practical instances including several examples, we refer the reader to [3].
In [28], an optimal inventory control policy for medicines with stochastic demands was created. The optimal policy determines the order quantity for each medicine for each time step that minimises the expected total inventory costs. In [15], a model was created for the smart home energy management system. The goal is to minimise the costs for supplying power and extracting power from the grid for a residence. The idea is to balance out the production and the expenditure of energy from devices at home. This can be achieved by saving energy in a battery and use the energy at some point in the future. The model determines the optimal

policy for the usage of the battery that minimises the expected total costs. In [8], a model was designed to assist food banks with equal distribution of different kind of supplies. Here they consider one warehouse, in which supplies arrive either by donations and transfers from other warehouses. Both supply routes are considered stochastic, and the demand is considered deterministic. The goal is to find a good allocation policy that equally distributes food to the people. Several allocation policies are tested and a description is given of the optimal policy. In [20] a policy is created for the frequency and duration for the follow-up of breast cancer patients. This policy is personalised for every patient, and several personal characteristics are considered, for example age of the patient. The policy determines when a patient should get a follow-up and when the patient should wait. The costs are a combination of the costs of the mammography and the life expectancy, which tries to detect issues as fast as possible while avoiding overtreatment.

Reinforcement learning is a field where MDP is used frequently. Reinforcement learning is concerned with learning what actions an agent should perform to minimise the costs or maximise the rewards. At the start the agent has no knowledge of which actions are good or bad. Iteratively the agent learns what actions to take in each state by using the observations made in previous iterations. For a complete introduction of the usage of MDP in reinforcement learning, we refer the reader to [37].

All current solution methods for obtaining the optimal policy for an MDP use the Bellman equations. Chapter 3 introduces the Bellman equations formally. The idea of the Bellman equations is to obtain a recurrent relation between states. This is then used to obtain the value of each state, where the value is equal to the instant reward plus the expected future reward, captured in the values of possible next states. Picking the decision in each state that maximises the expected reward determines the optimal policy.

Several exact algorithms that use the Bellman equations are backward dynamic programming, value iteration, policy iteration and modified policy iteration. Chapter 3 introduces theses algorithms. For a thorough treatment of these methods and the Bellman equations, we refer the reader to [24].

Most practical MDPs are too difficult to solve in a reasonable amount of computational time [18], [21]. The difficulties arise from the so-called curses of dimensionality. The three curses are size state space, size action set and amount of random possibilities, see [21]. These curses lead to the use of approximation methods to obtain an approximation of MDP. These approximation make use of the Bellman equations. Some examples of approximation methods for MDP are approximate dynamic programming (ADP), approximate policy iteration [7], approximate linear programming [29], and approximate modified policy iteration [30]. This thesis focuses on ADP, which is introduced in Chapter 3 and we will give a more extensive literature review in section 2.3.

2.2 Optimal stopping problem

Optimal stopping problems are concerned with choosing a time to stop a process, so that the costs are minimal. At every time step, the model is in a state. In this state, one can decide to

stop or to continue. When the process continues, a new state is determined randomly. This is repeated until the end horizon is reached, or when the process stops.
Time can be both discrete and continuous. Some practical examples of optimal stopping problems occur in gambling, house trading, and options pricing. For example, in the field of options pricing, the question resolves around when one should sell an option. The optimal policy to sell the option then determines the initial cost of an option. For an introduction to modelling of options pricing, we refer the reader to [32].
An optimal stopping problem can either be history-dependent or history-independent. If the optimal stopping is history-independent, then it can be formulated as searching for a stopping time in a Markov Chain [34]. A Markov Chain is a stochastic process that given a state randomly goes to a new state at every time step. The transition from state to state is history-independent and only depends on the current state. Markov Chains will be formally introduced in Chapter 3.
Optimal stopping problems are often considered history-dependent in the literature of options pricing [10]. Optimal stopping problems are difficult to solve optimally [10], hence approximation methods are used in most practical instances.
There are two commonly used types of solution methods for solving or approximating optimal stopping problems. The first approach is the dual approach, which carries some similarities to the dual approach of a linear program. The optimal stopping problem takes the view of the customer that seeks the optimal strategy to minimise its costs. In the dual approach, the problem is formulated that takes the view of the seller, where the maximum reward need to be determined, taking into consideration the constraints of the option. These two problems have different objective functions, where the optimums are equal. Instead of searching for a stopping time, the dual approach searches for an optimal martingale that corresponds to the costs. In [25] was the first occurrence of the dual method. Consequently, various algorithms have been created using this method. A study of the dual approach, including its analysis, can be found in [31], to which we refer the reader for a thorough treatment of the dual approach. For a more extensive literature view of the dual approach, we refer the reader to [10]. Chapter 4 will formally introduce a novel approach introduced in [10].
The second method to solve optimal stopping problems is ADP. A representation of optimal stopping problems with states and transitions can be made, making the Bellman equations use-able for optimal stopping.

2.3 Approximate dynamic programming

The idea of ADP is to use dynamic programming and the Bellman equations to approximate the value of each state. The values of each state are updated iteratively using the approximated values from the previous iteration. ADP goes forward in time, which makes is a forward dynamic programming method. The value of a state is updated by considering the instant costs plus expected future costs. The expected future costs are approximated using the approximated values from previous iteration. At every iteration an instance is created that determines which states are visited and updated. For a complete introduction to ADP, we

refer the reader to [21]. For a more practical view of the usage of ADP, we refer the reader to [18]. We will focus our attention on ADP in this thesis.

ADP is a method frequently used in practise to obtain a good policy. As stated in previous sections, ADP is used in both MDP and optimal stopping problems. We state several practical examples of ADP. In [13] a policy for patient admission planning in hospitals was created using ADP. The policy contains an admission and plan to allocate necessary resources to the patients. The model considers stochastic patient treatment and patient arrival rate. The state space considers multiple time periods, patient groups and different resources. The goals is to minimise the wait time for the patients with the given resources. In [17] a policy was obtained for the ambulance redeployment problem. In the ambulance redeployment problem the question is when to redeploy idle ambulance to maximise the amount of calls reached within a delay threshold. The state takes into consideration the state per ambulance (for example: idle, moving, on site), location of the ambulance and a queue of calls with priority and location that need to be answered. The objective is to minimise the amount of calls that are urgent that exceed the delay threshold. In [19] the single vehicle routing problem with stochastic demands was approximated using ADP. The single vehicle has to visit several customers with an uncertain demand until the vehicle reaches the customer. The vehicle has a maximum capacity and can restock its supplies at a depot. Travel costs are dependent on the distances between customers and the depot. The goal is to minimise the expected total travel costs. The policy consists of where the vehicle should drive in a given state, thus given the current capacity, location and customer demand, where should the vehicle go to? In [9] and [33] an policy was determined to play the game of Tetris. Tetris is a video game where the player has to place blocks in a grid. By making full rows in the grid, the row gets cleared and blocks are removed. The goal is to play for as long as possible. A more formal introduction will be given in section 3.5.1 and for a more complete overview of Tetris we refer the reader to [6]. In this thesis, we will approximate The Game [36] by using ADP, which has not been done before.

ADP is also applied on optimal stopping problems, specifically options pricing problems. The first occurrence of the usage of ADP in options pricing is [5]. Consequently in [16] and [34] approximation methods were created for history dependent optimal stopping time problems. Both methods use the basis functions approach to approximate the value of a state. The idea of basis functions is to use characteristics of a state to determine the value of a state. For a more formal definition of basis function, we refer the reader to [21]. We will also give a definition of basis functions in 3.4.2.

One important question stated in Chapter 1 is about the quality of ADP. Since approximation methods are used, one is interested in knowing whether the policy obtained is any good. We will discuss this question throughout the thesis in depth. Additional information can also be found in [21].

2.4 MDP as optimal stopping problem and contribution

In this section, we state several models and briefly explain them. Consequently we link the results of the thesis to these models and methods to show the contribution of this work. The idea of the model, denoted by OS-MDP, is to define a stochastic process over the policy space. The goal is to find a stopping time that minimises the expected total costs obtained by the current policy. A formal definition of OS-MDP will be given in Chapter 5.

In [12] an constrained MDP model was created that implements a stopping time. Additional to the normal set of actions, a terminate action is added that terminates the process. The total costs are the sum of all previously obtained costs up till termination. After termination, no more actions and costs are added. Hence a optimal stopping problem goes side by side the MDP process, both run over the time. This is different from OS-MDP since the optimal stopping problem goes on the policy space and the MDP goes through time.

A method that also searches in the policy space for a good policy is simulated annealing, as well as other heuristic methods. At every iteration, a neighborhood of the policy is defined from which the next policy is taken. This neighborhood consists of a set of policies which contain both better and worse policies. The policy taken from the neighborhood is then accepted with a certain probability, otherwise it remains at the current policy. This is repeated until some stopping criteria is reached, usually using a cooling scheme. The idea of a cooling scheme is to reduce the probability of accepting a policy that is very different from the current one as time progresses. Hence when more iterations are performed and the algorithm 'cools down', there will be less exploration and more exploitation. For a more extensive overview of simulated annealing we refer the reader to [1].

The difference between simulated annealing and OS-MDP is that OS-MDP seeks for a stopping time, which simulated annealing does not.

The contribution of this thesis is a model that uses the policy space of an MDP and searches for the optimal stopping time and its corresponding value. The idea is to define a Markov chain that goes from policy to policy. The objective is to find the optimal stopping time on which the Markov chain should stop. In other words, when is the policy found better then the policies you expect to find in the future. Additionally the different results of optimal stopping can be linked to solution methods of MDP, which we will discuss in Chapter 5.

Chapter 3

Markov Decision Process

A Markov decision process, or MDP for short, is a widely used formulation for di□erent problems. Chapter 2 lists various applications of MDP. This chapter gives a formal definition of MDP in section 3.2. Additionally, in section 3.3, several exact solution methods are introduced . We also introduce the curses of dimensionality, which are the reasons why MDPs are so challenging to solve. Section 3.4.1 discusses approximate dynamic programming, or ADP for short. Section 3.5 formulates two games, Tetris and The Game, as MDP.

3.1 Preliminaries

We denote the discrete set $[1, 2, \ldots, T]$ as $[1, T]$ throughout the thesis. We assume T to be finite. The norm of a vector v is defined as $||v|| = \sup_{s2S} |v(s)|$. The definition of a stochastic process is given [26].

Definition 1. *(Stochastic process) A stochastic process $\mathbf{Y} = \{Y_t, t \in [1, T]\}$ is a collection of random variables. That is, for each t in the set $[1, T]$, Y_t is a random variable.*

Any realization of $\mathbf{Y}$ is called a sample path. Define $Y_{[t]} = \{Y_1, Y_2, \ldots, Y_t\}$, thus the stochastic process until time t.
Let $g_t(Y_{[t]})$ be a payout function of $Y_{[t]}$ at time t. We assume that $g_t \in [0, 1]$. Note that any problem that does not satisfy these assumptions can be adjusted to satisfy the assumption. Next, the definition of a Markov Chain is given, taken from [24].

Definition 2. *(Markov chain) Let $\{Y_t, t \in [1, T]\}$ be a sequence of random variables which assume values in a discrete (finite or countable) state-space S. We say that $\{Y_t, t \in [1, T]\}$ is a Markov Chain if*

$$\mathsf{P}(Y_t = s_t | Y_{t\ \ 1} = s_{t\ \ 1}, \ldots, Y_0 = s_0) = \mathsf{P}(Y_t = s_t | Y_{t\ \ 1} = s_{t\ \ 1}), \tag{3.1}$$

for $t \in [1, T]$ and $s_t \in S$ for all $t \in [1, T]$.

Equation (3.1) is often called the Markov property, which states that given the current state the future is independent of the past.

A Markov chain can be stated as a 2-tuple (S, P), state-space S and transition matrix P. We extend the Markov chain by adding a cost function to a transition, therefore obtaining a 3-tuple (S, P, C_M). Let $MC = (S, P, C_M)$. The cost function $C_M(s_t, s_{t+1})$ can depend on both the state and the transition or either of them. Define

$$G(MC) = \mathsf{E}\Big[\sum_{t=0}^{T} \lambda^t C_M(s_t, s_{t+1})\Big], \tag{3.2}$$

which assigns a value to Markov Chain MC, where λ is the discount factor.

3.2 Definition Markov decision process

Markov Decision Processes (MDP) are discrete time multistage stochastic control problems. An MDP consists of a state-space S, where at each time step t the process is in a state s_t. At each time step, an action x_t can be taken from the feasible action set $X(s_t)$. Taking action x_t in state s_t results in a cost or reward, given by $C_t(s_t, x_t)$. The process then transitions to a new state s_{t+1} with probability $\mathsf{P}(s_{t+1}|s_t, x_t)$. Note that the transition probability of state s_t to s_{t+1} is independent of previously visited states and actions; this is called the Markov property. An MDP is often written as a 4-tuple: (S, X, P, C), where P is the transition function that determines the transitions probabilities. Let π be a policy, which is a decision function that assigns an action x_t to every state $s_t \in S$. Denote Π as the set of potential policies, thus $\pi \in \Pi$. Let $|\Pi|$ denote the cardinality of Π, which we assume to be finite. Note that if π is fixed, then the MDP becomes a Markov chain, see Definition 2.
Our goal is to find a policy that minimises the costs, which can be written as

$$\min_{\pi 2 \Pi} \mathsf{E}\Big[\sum_{t=0}^{T} \lambda^t C_t(s_t, x_t)\Big], \tag{3.3}$$

where λ is the discount factor and T is the length of the planning horizon. If $\lambda \in (0, 1)$, then we have a discounted MDP. When $\lambda = 1$ it is an expected total reward MDP. Suppose that π is an optimal policy. We assume that T is finite, which means the problem becomes a finite horizon MDP. Also, S is finite and, $X(s)$ is finite for all $s \in S$. The cost function C_t gives non-infinite costs. If the original problem is a maximisation problem, then a similar minimisation problem can be created. This can be done by multiplying all rewards by -1, turning the rewards into costs.
Given some MDP by (S, X, P, C) and some fixed policy π, then the transition probabilities P are fixed. Consequently, the MDP becomes a Markov Chain denoted by (S, P, C). Let $MC(\pi)$ be the Markov chain created by fixing policy π for the MDP.

3.3 Exact solution methods

Several methods exist that can solve MDPs. This section introduces the general dynamic programming approach. After that, four algorithms using this DP approach will be introduced: backward dynamic programming, policy iteration, value iteration and modified policy

iteration. Section 3.3.6 introduces the curses of dimensionality. These curses state the reasons why an MDP is challenging to solve.

3.3.1 General approach: dynamic programming

Dynamic programming is a possible solution method for solving MDPs, which splits the problem up in smaller sub-problems. The solutions of these sub-problems combined give the optimal solution for the MDP. Applying the Bellman equations [24] solve an MDP using dynamic programming. The Bellman equations are

$$V_t(s_t) = \min_{x_t 2 X_t} \Big(C_t(s_t, x_t) + \sum_{s^0 2 S} \lambda \mathsf{P}(s_{t+1} = s^0 | s_t, x_t) V_{t+1}(s^0) \Big), \quad \forall s_t \in S, \tag{3.4}$$

which computes the value $V_t(s_t)$ of being in state s_t. Let ω_{t+1} denote the random information that arrives after t, thus after action x_t is taken in state s_t. Therefore ω_{t+1} determines, given s_t, the next state s_{t+1}. Let Ω_{t+1} be the set of all possible ω_{t+1}. Define $S^M(s_t, x_t, \omega_{t+1}) = s_{t+1}$, which determines the state s_{t+1} given the previous state s_t, action taken x_t and random information ω_{t+1}. Equation (3.4) yields,

$$V_t(s_t) = \min_{x_t 2 X_t} \Big(C_t(s_t, x_t) + \sum_{\omega 2 \Omega_{t+1}} \lambda \mathsf{P}(W_{t+1} = \omega) V_{t+1}(s_{t+1} | s_t, x_t, \omega) \Big), \quad \forall s_t \in S. \tag{3.5}$$

Equation (3.5) is used when we refer to the Bellman equations for the rest of the thesis. The optimal policy consists of the best action in every state. Knowing the value of each state determines the optimal action by choosing the action with the lowest expected costs.

3.3.2 Backward dynamic programming

Backward dynamic programming (BDP) is a solution method for finite horizon MDPs. BDP goes backwards in time. Going backwards means that instead of starting at $t = 1$, it starts at $t = T$. The values of all states $s_T \in S_T$ are computed, hence computing $V_T(s_T)$ first. This computation is not complicated, because the expected future cost is zero and therefore, only the instant costs are relevant. After that, a step backwards is taken, going to $t = T - 1$. Computing the value $V_{T\ 1}(s_{T\ 1})$ is then done by using $V_T(s_T)$, the expected future costs. This process repeats until $t = 1$ is reached. The values of the states are then optimal, and therefore, the optimal policy can then be computed.
Algorithm 1 gives BDP in pseudo-code. It might not be possible to list all possible states s_T because they are unknown or there are too many. In that case, BDP does not work and a di□erent method needs to be used.

3.3.3 Value iteration

Value iteration (VI) is an algorithm that iteratively computes the value of each state until some stopping criteria is satisfied. We refer the reader to [24] for a complete overview of value iteration. This section gives a summary of the method and results.
When value iteration terminates, then a policy is obtained by taking the best action in

Algorithm 1: Backward dynamic programming for finite horizon MDP

Result: Optimal policy for finite horizon MDP
Set $t = T$;
Set $V_t(s_t) = C_t(s_t)$ for all $s_t \in S$;
while $t \neq 1$ do
  t=t-1 ;
  for $s_t \in S$ do

$$V(s_t) = \min_{x2X_t} \Big(C_t(s_t, x_t) + \sum_{\omega 2 \Omega_{t+1}} \mathsf{P}(W_{t+1} = \omega) V_{t+1}(s_{t+1} | s_t, x_t, \omega) \Big) \tag{3.6}$$

  end
  Set

$$X_{s_t,t} = \arg\min_{x2X_t} \Big(C_t(s_t, x_t) + \sum_{\omega 2 \Omega_{t+1}} \mathsf{P}(W_{t+1} = \omega) V_{t+1}(s_{t+1} | s_t, x_t, \omega) \Big) \tag{3.7}$$

end

each state. The stopping criterion is necessary, because even though the value of each state converges, the values might never reach the limit. The stopping criterion relies on some small parameter $\gamma > 0$. Definition 3 states the definition of a γ-optimal policy.

Definition 3. *(γ-optimal) Let $\gamma > 0$ and denote V_π as the vector that contains the value of each state given policy π. A policy π_γ is γ-optimal if for all $s \in S$,*

$$V_{\pi_\gamma} \geq V_\pi \ - \gamma. \tag{3.8}$$

VI finds an γ-optimal policy. Denote V^n as the value of each state in vector form at iteration n. Algorithm 2 gives the basic value iteration algorithm. Define $\Pi^n_{VI} = \arg\min_{\pi 2 \Pi}(C_\pi + \lambda P_\pi V^n)$ as all policies corresponding to value vector V^n. Next, we state several essential results of value iteration concerning the convergence of the algorithm.

Theorem 1. *Let $\{V^n\}$ be the values of value iteration for $n \geq 1$ and let $\gamma > 0$. Then the following statements about the value iteration algorithm hold:*

1. *V^n converges in norm to V ,*
2. *the policy π_γ is γ-optimal,*
3. *the algorithm terminates in finite N,*
4. *it converges $O(\lambda^n)$.*
5. *for any $\pi^n \in \Pi^n_{VI}$,*

$$||V_{\pi^n} - V_\pi \ || \leq \frac{2\lambda^n}{1-\lambda} ||V^1 - V^0||. \tag{3.11}$$

Generally, when λ is close to 1, the algorithm converges slowly. The convergence speed can be improved by several improvements and variants of value iteration. We will not discuss these variants and instead refer the reader to [24].

Algorithm 2: Basic value iteration algorithm

Result: γ-optimal strategy π_γ and value of MDP
Initialise V^0, $\gamma > 0$ and $n = 0$. ;
while $||V^{n+1} - V^n|| \geq \gamma(1-\lambda)/2\lambda$ *or* $n = 0$ do
For each $s \in S$, compute $V^{n+1}(s)$ using

$$V^{n+1}(s) = \min_{x2X(s)} \Big(C(s,x) + \sum_{s^0 2S} \lambda \mathsf{P}(s^0|s,x) V^n(s^0) \Big). \tag{3.9}$$

Increment n by 1.
end
For each $s \in S$ choose

$$\pi_\gamma(s) \in \arg \min_{x2X(s)} \Big(C(s,x) + \sum_{s^0 2S} \lambda \mathsf{P}(s^0|s,x) V^n(j) \Big). \tag{3.10}$$

3.3.4 Policy iteration

Policy iteration (PI) is an algorithm that computes the optimal policy π as well as the corresponding value of each state. For a complete overview of PI, we refer the reader to [24]. This section summarizes policy iteration.
First, some preliminaries. Let P_π be the transition matrix of the MDP under policy π. Let π^n denote the policy found in iteration $n \in [1, N]$. Define C_π as the vector of costs of every state given policy π: a vector of length $|S|$ containing $C_\pi(s)$ for all $s \in S$. Let V_π also be a vector of length $|S|$ containing the value of each state under policy π and V^n be the vector of values of states at iteration n. The matrix I denotes the identity matrix. Algorithm 3 gives the policy iteration algorithm. Define the set $\Pi^n_{PI} = \arg\min_{\pi 2 \Pi}\{C_\pi + P_\pi V^n\}$ and additionally $\pi \in \Pi^n_{PI}$

Algorithm 3: Policy iteration for infinite horizon MDP

Result: Optimal strategy π and value of MDP
Select an arbitrary policy $\pi^0 \in \Pi$ and set $n = 0$;
while $\pi^n \neq \pi^{n\ 1}$ *or* $n = 0$ do
Obtain V^n by solving

$$(I - \lambda \mathsf{P}_{\pi^n}) V^n = C_{\pi^n}. \tag{3.12}$$

Choose

$$\pi^{n+1} \in \arg \min_{\pi 2 \Pi} \{C_\pi + P_\pi V^n\}, \tag{3.13}$$

setting $\pi^{n+1}(s) = \pi^n(s)$ whenever possible. Increment n by 1.
end
Set $\pi = \pi^n$

implies that $\pi(s) = \pi^n(s)$ is set as often as possible. Hence the set Π^n_{PI} contains the set of best-improving policies for π^n that are as similar as possible. Next, an important result of the policy iteration algorithm, which relates the successive values V^n to V^{n+1}.

Theorem 2. *Let V^n and V^{n+1} be successive values of the policy iteration. Then $V^{n+1} \leq V^n$.*

Theorem 2 implies that in every iteration V^n improves or stays equal. Assume finite costs, finite state-space and finite action set. This assumption implies that V^n will converge to some value V . This value will be the optimal value, which gives us the optimal policy π .
The following theorem provides conditions for which the convergence is quadratic.

Theorem 3. *Suppose $\{V^n, n \geq 1\}$ is generated by policy iteration and $\pi^n \in \Pi$ for each n and there exists a K, $0 < K < \infty$ for which*

$$||P_{\pi^n} - P_\pi \,|| \leq K||V^n - V \,||, \tag{3.14}$$

for $n = 1, 2, \ldots$. Then

$$||V^{n+1} - V \,|| \leq \frac{K\lambda}{1-\lambda}||V^n - V \,||^2. \tag{3.15}$$

3.3.5 Modified policy iteration

Modified policy iteration (MPI) is an algorithm that combines both value iteration and policy iteration, introduced in sections 3.3.3 and 3.3.4, respectively. The idea of MPI is to execute PI, and after every policy improvement step, perform several VI steps. This process repeats until some stopping criterion is satisfied, which is similar to the stopping criteria of value iteration. The algorithm gives an γ-optimal policy. This section gives a formal definition of modified policy iteration.
Define $\{m_n, n \geq 1\}$ as a sequence of non-negative integers, called the order sequence. The order sequence determines the number of partial policy evaluations (or value iteration steps) done per policy improvement. Algorithm 4 gives the modified policy iteration algorithm.

Theorem 4 is a theorem related to the convergence of modified policy iteration.

Theorem 4. *Suppose V^0 initial value of modified policy iteration. Then, for any order sequence $\{m_n, n \geq 1\}$,*

1. *the iterations of modified policy iteration $\{V^n\}$ converge monotonically and in norm to V_λ, and*
2. *the algorithm terminates in a finite number of iteration with an γ-optimal policy.*

Deciding on the order sequence $\{m_n, n \geq 1\}$ is an interesting topic. Theorem 4 states that convergence of MPI is achieved for any m_n. The convergence speed does depend on m_n. The next corollary state the convergence rate for modified policy iteration.

Corollary 1. *Suppose $V^0 \geq 0$ and $\{V^n\}$ is generated by modified policy iteration, that π^n is a V^n-improving decision rule and π is a v_λ-improving decision rule. If*

$$\lim_{n \,!\, 1} ||P_{\pi^n} - P_\pi \,|| = 0, \tag{3.19}$$

then, for any $\gamma > 0$, there exists an N for which

$$||V^{n+1} - V_\lambda \,|| \leq (\lambda^{m_n+1} + \gamma)||V^n - V_\lambda \,|| \tag{3.20}$$

for all $n \geq N$.

Algorithm 4: Modified policy iteration

Result: γ-optimal strategy π_γ
Select a V^0, specify $\gamma > 0$ and set $n = 0$;

1. (Policy improvement) Choose π^{n+1} to satisfy

$$\pi^{n+1} \in \arg\min_{\pi 2 \Pi}\{C_\pi + P_\pi V^n\}, \tag{3.16}$$

   setting $\pi^{n+1} = \pi^n$ if possible. ;

2. (Partial policy evaluation)

   (a) Set $k = 0$ and compute

$$u_n^0 = \min_{\pi 2 \Pi}(C_\pi + \lambda P_\pi V^n). \tag{3.17}$$

   (b) If $||u_n^0 - V^n|| < \gamma(1-\lambda)/2\lambda$, go to step 3. Otherwise continue.

   (c) If $k = m_n$, go to (e). Otherwise compute

$$u_n^{k+1} = C_{\pi^{n+1}} + \lambda P_{\pi^{n+1}} u_n^k. \tag{3.18}$$

   (d) increment k by 1 and go to (c).

   (e) Set $V^{n+1} = u_n^{m_n}$, increment n by 1 and go to step 2.

3. Set $\pi_\gamma = \pi^{n+1}$ and stop.

This corollary states that the rate of convergence is bounded by $m_n + 1$. Note that value iteration optimises over the entire policy space at every iteration. Modified policy iteration does not do this, but instead uses a single policy to evaluate the value of each state and updates the policy similar to policy iteration. This means that modified policy iteration has a better convergence rate than value iteration.

3.3.6 Curses of dimensionality

Solution methods that find the optimal policy for an MDP generally do not work in practice because of computational difficulties. These difficulties are called the *curses of dimensionality*, which are now briefly discussed.

The first difficulty is the size of the state-space. In problems with a large state-space, computing the value of every single state is hard. This difficulty is because equation 3.4 needs solving for every single state.

The second curse of dimensionality is the size of the action set. To find the optimal action in equation (3.4), possibly every action has to be checked to determine the optimal action. This is computational difficult when the action set in every state is large.

The third and last curse of dimensionality is the size of the outcome space. The set of different

random outcomes ω_t defines the outcome space in a state. Computing the value of the state requires a summation of $\omega \in \Omega_t$, which is computational demanding when the set Ω_t is large. Because of these three curses, practical situations often use approximation methods.

3.4 Approximate dynamic programming

This section discusses approximate dynamic programming (ADP). Section 3.4.1 gives a formal definition of ADP, including a pseudo-code. Section 3.4.2 explains several techniques that can help overcome the curses of dimensionality. Consequently, section 3.4.3 discusses several challenges of creating an ADP algorithm.

3.4.1 Definition approximate dynamic programming

Approximate dynamic programming (ADP) is a method to approximate the value of a state using dynamic programming. The approximations then determine the corresponding policy, which is not necessarily optimal. ADP goes forward in time, thus starting from $t = 1$ and going forward. The general idea of ADP is to take N iterations, and for each iteration, a sample path is used to update the value of being in a state. Let $\hat{\omega} = (\omega_1, \omega_2, \ldots, \omega_T)$ be a sample path containing all relevant random information. Let $\hat{\omega}^n$ be the sample path of iteration $n \in [1, N]$. Define s_t^n as the state at time t in iteration n and define $\overline{V}_t^n(s_t^n)$ as the approximate value of this state in iteration n. Let π^n be the policy found in iteration n. Then find $\pi^n(s_t^n)$ by computing

$$\pi^n(s_t^n) = \arg\min_{x_t^n 2 X_t} \Big(C_t(s_t^n, x_t^n) + \lambda \sum_{\omega 2 \Omega_{t+1}} \mathsf{P}(W_{t+1} = \omega) V_{t+1}(s_{t+1}^n | s_t^n, x_t^n, \omega) \Big), \tag{3.21}$$

where x_t^n is the action taken in state s_t^n. Let

$$\hat{v}_t^n = \min_{x_t 2 X_t} \Big(C_t(s_t^n, x_t^n) + \lambda \mathsf{E}[\overline{V}_{t+1}^{n\ 1}(s_{t+1}^n | s_t^n, x_t)] \Big) \tag{3.22}$$

be the value approximation of state s_t^n. Update the value V using

$$\overline{V}_t^n(s_t^n) = (1 - \alpha_{n\ \ 1}) \overline{V}_t^{n\ \ 1}(s_t^n) + \alpha_{n\ \ 1} \hat{v}_t^n, \tag{3.23}$$

where α_n is a scalar dependent on the iteration. Section 3.4.3 gives more in-depth information on α_n. Algorithm 5 gives the basic ADP method, using equation (3.23), as pseudo-code.

3.4.2 Techniques for ADP

There are several techniques that can be applied to ADP to improve it or give ways to deal with the curses of dimensionality. This section discusses the following techniques: post-decision state, state aggregation and basis functions.

Post-decision state

Introducing post-decision states give us a way to deal with the large outcome space, which is one of the curses of dimensionality. The post-decision state is a state after action x_t at

Algorithm 5: Basic ADP algorithm

Result: Approximation of V ;
Initialize $\overline{V}_t^0(s_t)$ for all states s_t ;
Choose initial state s_0^1 ;
for $n = 1$ *to* N do
  Choose sample path $\hat{\omega}^n$;
  for $t = 0, 1, \ldots, T$ do
    Solve

$$\hat{v}_t^n = \min_{x_t^n 2 X} \Big(C_t(s_t^n, x_t^n) + \lambda \mathsf{E}[\overline{V}_{t+1}^{n\ 1}(s_{t+1}^n | s_t^n, x_t^n)] \Big); \tag{3.24}$$

    and

$$\hat{x}_t^n = \arg\min_{x_t^n 2 X} \Big(C_t(s_t^n, x_t^n) + \lambda \mathsf{E}[\overline{V}_{t+1}^{n\ 1}(s_{t+1}^n | s_t^n, x_t^n)] \Big); \tag{3.25}$$

    Update $\overline{V}_t^{n\ \ 1}(s_t)$ using

$$\overline{V}_t^n(s_t) = \begin{cases} (1 - \alpha_{n\ \ 1})\overline{V}_t^{n\ \ 1}(s_t^n) + \alpha_{n\ \ 1}\hat{v}_t^n & s_t = s_t^n; \\ \overline{V}_t^{n\ \ 1}(s_t) & \text{otherwise;} \end{cases} \tag{3.26}$$

    Compute $s_{t+1}^n = S^M(s_t^n, x_t, \hat{\omega}^n(\omega_{t+1}))$;
  end
end

state s_t but before any new information arrives (ω_{t+1}). Therefore not all outcomes of ω need consideration for every action.

Let s_t^x denote the post-decision state directly after taking action x_t in state s_t. Let the function $S^{M,x}(s_t, x_t) = s_t^x$ output the post-decision state. The variable $V_t^x(s_t^x)$ gives the value of a post-decision state, and $\overline{V}_t^x(s_t^x)$ its approximation. The value of a post-decision state is given by $V_t^x(s_t^x)$ and its approximation is $\overline{V}_t^x(s_t^x)$. Compute the value of a post-decision state by

$$V_t^x(s_t^x) = \mathsf{E}[V_{t+1}(s_{t+1} | s_t^x, \omega)]. \tag{3.27}$$

Computing the value this way means we do not have to evaluate the di□erent options of ω for every action $x_t \in X_t$ for every state s_t. In addition to equation (3.27), an update of $V_t(s_t)$ is also required, given by

$$V_t(s_t) = \min_{x_t 2 X_t} \Big(C_t(s_t, x_t) + \lambda V_t^x(s_t^x) \Big). \tag{3.28}$$

Note that combining equations (3.27) and (3.28) obtains the Bellman equations.

Algorithm 5 needs some modifications to use post-decision states. Equation (3.24) changes to

$$\hat{v}_t^n = \min_{x_t^n 2 X} \Big(C_t(s_t^n, x_t^n) + \lambda \overline{V}_t^x(s_t^x) \Big) \tag{3.29}$$

and equation (3.25) changes to

$$\hat{x}_t^n = \arg\min_{x_t^n 2 X} \Big(C_t(s_t^n, x_t^n) + \lambda \overline{V}_t^x(s_t^x) \Big). \tag{3.30}$$

Updating $\overline{V}_t^{x,n}$ can be done in several ways. We state one example [22]:

$$\overline{V}_{t\ 1}^{x,n}(s_{t\ 1}^{x,n}) = (1-\alpha_{n\ \ 1})\overline{V}_{t\ 1}^{x,n\ \ 1}(s_{t\ 1}^{x,n}) + \alpha_{n\ \ 1}\hat{v}_t^n. \tag{3.31}$$

State aggregation

One of the curses of dimensionality discussed in section 3.3.6 is the size of the state-space. A possible way to overcome this is by aggregating the state-space. Aggregation means that states get grouped, and only the new grouped states get considered. This aggregation reduces the size of the state-space used. The aggregation depends highly on the problem. Usually, states are grouped together based on characteristics that the states have in common.
It is possible to apply multiple levels of aggregations. For example, consider locations in a city. Every location has some coordinate and naturally groups into some street, area, city, region and country. Every step is an additional level of aggregation. Considering every single location is very di□cult, but considering a set of cities is do-able.
We will now define state aggregation formally. Define the function $\mathcal{G}^g : S \to S^g$, where g stands for the level of aggregation. Let $s^g = \mathcal{G}^g(s)$ be the gth aggregation of state s. An important constraint on the aggregation is that every state $s \in S$ belongs to some aggregated state for every aggregation level g. Let G be the set of all aggregation levels; therefore, $g \in G$. Define w^g as weight function, dependent on the aggregation. Computing the approximate value of a state is done by

$$\overline{V}(s) = \sum_{g2G} w^g \overline{V}^g(s), \tag{3.32}$$

where $\overline{V}^G(s)$ is the approximated value of the aggregated state. The weight w^g is updated every iteration; thus $w^{(g,n)}$ is used instead. For a complete overview of the usage of state aggregation, we refer the reader to [21].

Basis functions

Basis functions is a commonly used strategy for ADP. The idea is to capture elements of a state to compute the value of a state. These elements will be called features, denoted by f and let F be the set of all features. The basis function then computes the value of a specific feature, given by $\phi_f(s)$ for state s and $f \in F$. The approximated value of a state is then computed by

$$\overline{V}(s_t|\theta) = \sum_{f2F} \theta_f \phi_f(s_t), \tag{3.33}$$

where θ_f is a weight factor corresponding to feature f. Note that this is a linear function, but the basis functions ϕ_f do not have to be linear. The weight factor θ_f updates at every iteration and time step, hence depends on n. Therefore we write θ_f^n.
Basis functions are often implemented simultaneously with post-decision states. Equation (3.33) obtains a value for the post-decision state. The computation of $\hat{v}_t^n$ is done by

$$\hat{v}_t^n = \min_{x_t 2X} \Big(C_t(s_t^n, x_t) + \sum_{f2F} \theta_f^n \phi_f(s_t^{x,n}) \Big). \tag{3.34}$$

Note that basis functions can be applied in combination with both state aggregation and post-decision states.
There are several methods to update θ_f^n. The appendix contains the recursive least squares method for updating θ_f^n [18]. We apply the recursive least squares method in section 3.5.3 to The Game.

3.4.3 Challenges of ADP

Step size function

Equation (3.23) introduces the factor α_n. The factor $\alpha_n \in [0,1]$ assigns a priority to new values. If $\alpha_n = 0$ for all n, then the value of a state is never updated. The challenge is to pick a sequence α_n that guarantees convergence of $\overline{V}(s)$. There are di□erent update functions used for α_n [23]. Note that using basis functions makes the stepsize function unnecessary.

Exploration versus exploitation

Algorithm 5 chooses a sample path $\hat{\omega}^n$ every iteration. Deciding on a sample path is a challenge of its own. For example, when the best state is always visited, a problem occurs. When the value of a state gets updated, usually it increases (or in other settings, decreases), making it more likely to get picked in the sample path. Hence exploitation will be done, but the same set of states will always be visited, meaning no exploration.
Generating a completely random sample path leads to exploration. However, the values of the states will not be accurate, since they rarely update. Hence a balance between exploration and exploitation is needed. A simple way to do this is at every time step of a sample path is generated: flip a coin. Either the 'best' state is picked or some random state. However, a single step of exploration does not achieve much.
Therefore when deciding on a sample path, we need a balance between exploration and exploitation [21]. This is problem dependent since it depends on the size of the state-spaces and action set.

Evaluating ADP

Using an approximation method raises a question about the quality [23]. Several methods exist that evaluate the quality of approximations. Three methods are discussed.
The first way is to compare the solution of ADP with the optimal solution. A exact MDP solution method is generally unfit for complex problems. However, MDP is usable for smaller problems. Hence, computing solutions for a small MDP allows a comparison between the exact solution and ADP solutions.
The second method is to compare ADP to other solution methods. ADP's are comparable, which indicates the quality of the chosen ADP. Another possibility is to use other approximation methods as comparison tools, for example, simulation methods.
The latter approach is by using an approximate guarantee, which a relation between the approximation and the optimum. For example, in a minimisation setting $OPT \leq APPROX$.

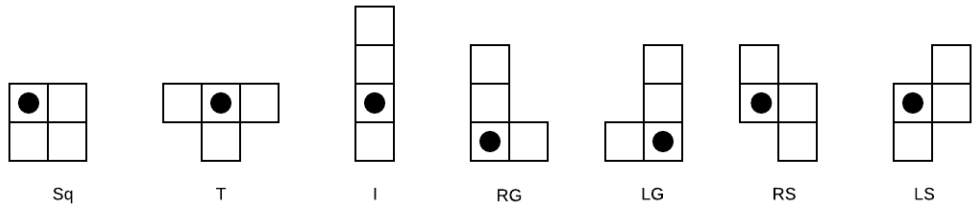

Figure 3.1: All tetrominoes with their corresponding initials.

This is a trivial bound, but it is an approximate guarantee nonetheless. In the subject of scheduling problems, various approximate guarantees are known for algorithms. In the case of ADP, though, there are no general approximate guarantees.

3.5 MDP examples

The upcoming sections discuss Tetris and The Game. Tetris is an infinite horizon MDP. Tetris is a well-known benchmark problem, so there is plenty of literature available. This thesis discusses Tetris due to its benchmark status. The Game is a card game developed by White Goblin Games [36] playable by 1 to 5 players. This thesis discusses The Game because it is a finite horizon MDP with a large state-space, large action set and a large variety of random arrivals. This means that BDP approaches for MDP do not work because of the curses of dimensionality. Therefore, we discuss how to apply ADP. Section 3.5.1 introduces Tetris and section 3.5.2 introduces and models The Game. Section 3.5.3 defines ADP for The Game.

3.5.1 Tetris

Tetris is a puzzle video game[6]. There are several definitions on how Tetris is played, differing for example in the arrival of the tetrominoes. Therefore, explaining the rules of Tetris comes first.

Tetris is played on a 10 by 20 grid, where every point in the grid is either full or empty. When a row consists of only full cells, the row is cleared; becomes empty and everything above it moves one row down. There are seven different tetrominoes that can arrive, all with equal probability. When playing, only the current tetromino is known. When placing a block, it falls from the top to the bottom, and can move horizontally and rotate.

Tetris ends when the block that needs placing can no longer be placed in the 10 by 20 grid by a valid move. This is different from the original Tetris game, where the player has to survive for a predetermined amount of turns [6].

MDP formulation

Let t denote the time. There are seven di□erent tetrominoes in total. The di□erent tetrominoes are shown in figure 3.1. Let $\mathcal{D}$ denote the set of all tetrominoes, and let $b_t \in \mathcal{D}$ denote the tetromino that needs placing at time t. Let $\mathcal{H}$ be a 10×20 matrix containing zeros and ones. A one corresponds with a full cell, and a zero corresponds with an empty cell. None of the tetrominoes parts can cover an already full cell. Define $c \in [1, 10]$ as the column of the grid and $r \in [1, 20]$ as the row of the grid. For example, $\mathcal{H}(c, r)$ corresponds to the cell in column c and row r. Let $\mathcal{H}(:, r)$ corresponds to the entire row r, which is a vector with 10 elements and $\mathcal{H}(c, :)$ denotes column c. Then the state at time t is defined as

$$s_t = \{\mathcal{H}_t, b_t\} \tag{3.35}$$

and let S be the set of all possible states.
Note that in figure 3.1, there is a dot in every tetromino. Whenever a block is placed in column $c \in [1, 10]$, it means that the block with the dot is in column c. This placement rule means that not every orientation of a block is possible in every column. Define $\mathcal{C}_t \in [1, 10]$ as the decision to place the tetromino at time t in column $\mathcal{C}_t$. Let $\mathcal{R}_t$ be the decision on how to rotate b_t, where $\mathcal{R}_t(b_t) \in [0, 90, 180, 270]$. Depending on b_t, the decisions $\mathcal{R}_t$ and $\mathcal{C}_t$ are constrained. Duplicate rotations, yielding the same shape, are excluded. For example with SQ, which always has the same shape for every rotation. The constraint to $\mathcal{R}_t$ and $\mathcal{C}_t$ per tetromino are as follows:

$$SQ \quad \mathcal{R}_t = 0, \mathcal{C}_t \in [1, 10]$$

$$I \quad \begin{cases} \text{if } \mathcal{R}_t = 0, & \mathcal{C}_t \in [1, 10], \\ \text{if } \mathcal{R}_t = 90, & \mathcal{C}_t \in [2, 8]. \end{cases}$$

$$T \quad \begin{cases} \text{if } \mathcal{R}_t = 0 \text{ or } 180, & \mathcal{C}_t \in [2, 9], \\ \text{if } \mathcal{R}_t = 90, & \mathcal{C}_t \in [2, 10], \\ \text{if } \mathcal{R}_t = 270, & \mathcal{C}_t \in [1, 9]. \end{cases}$$

$$LS \quad \begin{cases} \text{if } \mathcal{R}_t = 0, & \mathcal{C}_t \in [1, 9], \\ \text{if } \mathcal{R}_t = 90, & \mathcal{C}_t \in [2, 9]. \end{cases}$$

$$RS \quad \begin{cases} \text{if } \mathcal{R}_t = 0, & \mathcal{C}_t \in [1, 9], \\ \text{if } \mathcal{R}_t = 90, & \mathcal{C}_t \in [2, 9]. \end{cases}$$

$$LG \quad \begin{cases} \text{if } \mathcal{R}_t = 0, & \mathcal{C}_t \in [2, 10], \\ \text{if } \mathcal{R}_t = 90, & \mathcal{C}_t \in [1, 8], \\ \text{if } \mathcal{R}_t = 180, & \mathcal{C}_t \in [1, 9], \\ \text{if } \mathcal{R}_t = 270, & \mathcal{C}_t \in [3, 10]. \end{cases}$$

$$RG \quad \begin{cases} \text{if } \mathcal{R}_t = 0, & \mathcal{C}_t \in [1, 9], \\ \text{if } \mathcal{R}_t = 90, & \mathcal{C}_t \in [1, 8], \\ \text{if } \mathcal{R}_t = 180, & \mathcal{C}_t \in [2, 10], \\ \text{if } \mathcal{R}_t = 270, & \mathcal{C}_t \in [3, 10]. \end{cases}$$

The decision x_t is defined as

$$x_t = (\mathcal{C}_t, \mathcal{R}_t). \tag{3.36}$$

Define the function $S^M(s_t, x_t, \omega_{t+1})$m which determines the state s_{t+1}. First $\omega_{t+1} \in \mathcal{D}$ gives a random block, hence $b_{t+1} = \omega_{t+1}$. Second, we give the transition from $\mathcal{H}_t$ to $\mathcal{H}_{t+1}$ dependent on b_t and x_t. Define function D as the function that removes any full rows in $\mathcal{H}$ and adds a zero row at the top. Define $h_c = \arg\max_{r2[1,20]}(r \times \mathcal{H}_t)$, which corresponds to the height of

column c. Let $\hat{\mathcal{H}}_t(b_t, x_t, \mathcal{H}_t)$ denote the zero matrix of 10×20 with 1 entries on the cells of block b_t's placement. The appendix includes a full overview of $\hat{\mathcal{H}}$. Define

$$\mathcal{H}_{t+1} = S^M(s_t, x_t, \omega_{t+1}) = D(\mathcal{H}_t + \hat{\mathcal{H}}_t). \tag{3.37}$$

The goal of Tetris is to survive for as long as possible. By placing tetrominoes such that rows fill, which remove themselves, one can achieve survival. Define $C(s_t, x_t) = 1$, whenever a tetrominoes placement is successful, and 0 otherwise. Hence the goal is to compute

$$\max_{\pi 2 \Pi} \mathsf{E}\Big[\sum_{t=0}^{1} \lambda^t C(s_t, x_t)\Big]. \tag{3.38}$$

3.5.2 The Game

The Game is a co-op card game made by White Goblin Games,played with 1 to 5 players [36]. This thesis considers the one player variant of The Game. The rules come first and after that an MDP formulation.

Rules

The Game consists of 102 cards, each having a number. There are two cards with the number 1, two cards with the number 100 and the remaining 98 cards, numbered 2 to 99, and are therefore unique. The 1's and 100's start open as four separate piles where cards are playable on. The remaining cards are shu□ed, and the player draws seven cards. Play starts from here.
The goal is to play as many cards as possible. Playing all the cards means a perfect score. The players turn consists of two phases:

1. Play at least two cards. If you cannot play two cards, then the game ends. When possible, you can still play one card.

2. Refill hand to seven cards or until the draw pile is empty.

Cards are played on one of the four piles. Throughout the game, there will be four piles where the player can play cards. On the two piles that started with a 1, only cards with a higher number then the previous card played on that pile is playable. On the two piles that start with 100 it is the other way around, so only cards with a lower number then the previous card can be played. There is one exception: a card with a di□erence of exactly 10 in the other direction. This is called *jumping back*. For example, on a 1 pile with 45, 35 is playable, but 44 is not.
When the drawing pile is empty, the player plays as many cards as possible. The player has a 'perfect memory', meaning that the player knows what have been played previously, what the cards in hand are and also what cards are in the drawing pile. The sequence of the drawing pile is unknown the player. Figure 3.2 gives a possible initial setup, the only thing being random are the cards in hand.

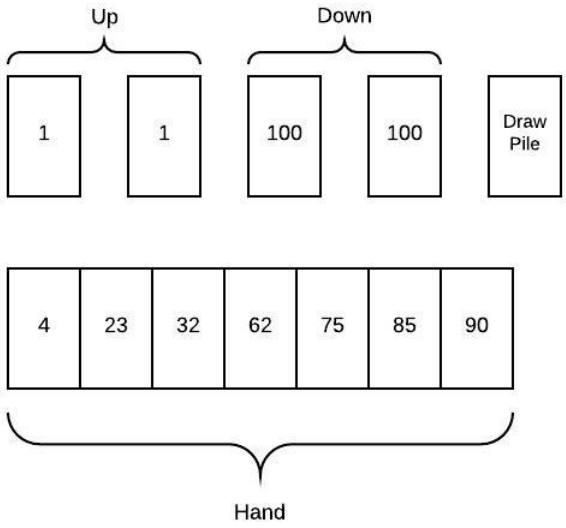

Figure 3.2: Example initial setup for The Game

MDP formulation

The planning horizon length T can be bounded by 49 since at least two cards are played each time step, and there are 98 cards in total. It is possible to finish earlier. This occurs either by playing all cards or by not being able to play two cards in one turn. Let $t \in [1, T]$ be the current time step.

To determine whether a card is playable on a pile, introduce the matrices $\mathcal{N}$ and $\mathcal{K}$. Let $\mathcal{N}$ be a 99×99 with 1 entries in the lower triangular part and 0 entries on the diagonal and upper triangular part. Let $\mathcal{K}$ be a 99×99 zero matrix with 1 entries on the $(i, i+10)$ entries, where $i = 1...89$. For cards played normally, use matrix $\mathcal{N}$. For cards played using the jump back move, use matrix $\mathcal{K}$.Define $\mathcal{L} = \mathcal{N} + \mathcal{K}$, which is a matrix used to determine whether a card can be played on a pile. Let h be a zero vector of length 99, with entry $h(c) = 1$ for $c \in [2, 99]$. Then, for a 1 pile, $\mathcal{L}h$ outputs a vector of length 99, with 1 entries on cards playable on card c. For a 100 pile, $\mathcal{L}^T$h outputs the vector, where $\mathcal{L}^T$ is the transposed matrix of $\mathcal{L}$.

Define Q_t as a vector containing the numbers of the four piles at time t. Let $Q_0 = [1, 1, 100, 100]$. Let H_t be a vector of length 99 containing ones and zeros. If a card $c \in [2, 99]$ is in the player's hand at time t, then $H_t(c) = 1$ and is 0 otherwise. Define K_t similar to H_t, except it contains information about the cards in the drawing pile. Thus $K_t(c) = 1$ implies c is in the drawing pile and 0 otherwise. Note that $H_t(c) = K_t(c) = 0$ implies that card c has been played. The state at time t can then be defined as

$$s_t = \{Q_t, H_t, K_t\}, \tag{3.39}$$

and S contains all possible combinations of s_t. The action x_t determines which card is played on which pile and let X be the action set. Let x_t be a matrix of 4 by 99, where $p \in [1, 4]$ denotes

the pile and $c \in [2, 99]$ denotes the card. Therefore $x_t(p, c) = 1$ implies card c is played on pile p at time t and 0 otherwise. Some constraints are required on x_t in a given state s_t. First $\sum_{p=1}^{4} \sum_{c=2}^{99} x_t(p, c) \geq 2$, which implies that at least two cards are played. Second $x_t \leq H_t$, which implies only cards in the current hand are played. Third $\sum_{p=1}^{4} x_t(p, c) \leq 1 \forall c \in [2, 99]$, which implies every card is only played once.

The action x_t is not enough to determine the next state. For example, consider the action to play 7, 13 and 17 on a 1 pile with number 8. There are two ways to play these cards, option 1: $13 \rightarrow 17 \rightarrow 7$ and option 2: $17 \rightarrow 7 \rightarrow 13$. Intuitively, option 1 looks better because it obtains a lower number. But if number 3 is still in the drawing pile, then option 2 can be better because of the jump back action $13 \rightarrow 3$. Therefore x_t needs to be extended. Define x_t^e as a matrix of 4 by 99, where $x_t^e(p, c) = 1$ implies that c is the number on pile p and 0 otherwise. The constraint $x_t^e(p, c) \leq x_t(p, c)$ is required, to make sure only played cards are usable. Let X^e be the set containing all possible x_t^e.

Combining all actions in every state gives a policy $\pi \in \Pi$. The goal is to maximise the amount of cards played. Therefore the reward function $C(s_t, x_t)$ is equivalent to the amount of cards played in that turn, thus $C(s_t, x_t) = \sum_{p=1}^{4} \sum_{c=2}^{99} x_t(p, c)$. The goal then becomes

$$OPT_{TG}^{\emptyset} = \max_{\pi 2 \Pi} \mathsf{E}\Big[\sum_{t=1}^{T} \sum_{c=2}^{99} \sum_{p=1}^{4} x_t(p, c)\Big]. \tag{3.40}$$

Note that this problem transforms into a minimisation problem with the following goal function:

$$OPT_{TG} = \min_{\pi 2 \Pi} \Big(98 - \mathsf{E}\Big[\sum_{t=1}^{T} \sum_{c=2}^{99} \sum_{p=1}^{4} x_t(p, c)\Big]\Big). \tag{3.41}$$

Since the maximum score is 98, $OPT_{TG}^{\emptyset}$ is non-negative.

Denote W_{t+1} as a vector of length 99 with zero and one entries, where $W_{t+1}(c) = 1$ implies card c is drawn and 0 otherwise. The vector W_{t+1} then denotes the cards drawn after action x_t. The probability distribution of W_{t+1} is

$$\mathsf{P}(W_{t+1} = \omega) = \begin{cases} 1 & \text{if } \sum_{c=2}^{99} \Big(K_t(c) + H_t(c)\Big) \leq 6, \omega = K_t, \\ \frac{(\sum_{p=1}^{4} \sum_{c=2}^{99} x_t)! (\sum_{c=2}^{99} K_t \quad \sum_{p=1}^{4} \sum_{c=2}^{99} x_t)!}{(\sum_{c=2}^{99} K_t)!} & \text{if } \begin{cases} \sum_{c=2}^{99} \Big(K_t(c) + H_t(c)\Big) > 6, \omega \leq K_t, \\ \sum_{c=2}^{99} \omega = \sum_{c=2}^{99} x_t, \end{cases} \\ 0 & \text{otherwise.} \end{cases} \tag{3.42}$$

The constraint $\sum_{c=2}^{99} \Big(K_t(c) + H_t(c)\Big) \leq 6$ implies that there are six or fewer cards left in the game; therefore all remaining cards will be drawn, denoted by $\omega = K$. When more then six cards are remaining, $\omega \leq K$ implies only cards in the drawing pile can be drawn and $\sum_{c=2}^{99} \omega = \sum_{c=2}^{99} x_t$ makes sure the same amount of cards are drawn and played.

The transition function $S^M(s_t, x_t, W_{t+1})$ determines the next state s_{t+1} given the previous state s_t, cards played in turn $x_t(p, c)$, and new cards added to hand W_{t+1}. Define the update

function for Q, denoted by $\mathcal{U}$, as

$$Q_{t+1}(p) = \mathcal{U}(Q_t(p), x_t^e) = \begin{cases} c & x_t^e(c,p) = 1, \\ Q_t(p) & \text{otherwise.} \end{cases} \tag{3.43}$$

Then the transition function becomes

$$s_{t+1} = S^M(s_t, (x_t, x_t^e), W_{t+1}) = \{\mathcal{U}(Q_t, x_t^e), H + W_{t+1} - x_t, K - W_{t+1}\}. \tag{3.44}$$

Next, algorithm 6 gives a pseudo-code for playing The Game.

Algorithm 6: Pseudo-code for The Game

```
Result: Score=Amount of cards played
Initialization, t = 0 ;
Q_t = {1, 1, 100, 100} ;
K_t = contains all cards;
Draw Hand ;
while Game not ended do
    t=t+1 ;
    Determine action set X;
    if sum_{c=2}^{99} K_t(c) = 0 then
        Play as many cards as possible ;
        Game ends ;
    else
        if X = ∅ then
            Play 1 or 0 cards ;
            Game ends ;
        else
            Find x ∈ X and x^e ∈ X^e that maximises reward function ;
            Update Q_t to Q_{t+1} ;
            Draw cards and update H_t to H_{t+1} and K_t to K_{t+1} ;
        end
    end
end
Score= 98 − sum_{c=2}^{99} K − sum_{c=2}^{99} H;
```

3.5.3 Applying ADP on The Game

This section describes how to apply ADP on The Game. A description of how to create a sample path is included and how we dealt with exploration versus exploitation. After that, an overview of the basis functions approach is given.

Deciding on a sample path

The basic ADP algorithm generates a sample path ψ^n. Section 3.4.3 discusses the problem of exploration versus exploitation. Here we state the specifics for The Game.
So how do we generate a sample path for The Game? The algorithm starts with some initial state, which corresponds to some initial starting hand with no cards played yet. Then a sample path is a random sequence of cards of the drawing pile.
How should this random sequence be generated? Total random is a possibility, which implies much exploration. It is also possible to use the same sequence or make tiny changes, in which case there is exploitation but little exploration.
What is the 'best' can only be determent by trying different strategies and concluding which one is best. In the example of The Game, the following strategy will be applied:

1. Generate some random sequence.
2. Use this random sequence for N_1 iterations, possibly with some very small changes to the sequence. (Exploitation)
3. Go back to 1. Repeat N_2 times. (Exploration)

This requires $N_1 \times N_2$ iterations. To find the right balance between exploration and exploitation, experiment with different combinations. In Chapter 6, several different settings of N_1 and N_2 will be tested.

Value of state: basis function

The basis function approach, introduced in section 3.4.2, will be applied to approximate the value of the post-decision state. We will use the recursive least squares method [18] to update θ_f^n, which is also in the appendix. There are many possible features, we state several examples:

1. Playable space per pile, which implies 4 different features.

$$\phi_f(s_t) = 99 - Q_t(p) \quad \text{for } p = 1, 2 \text{ and} \qquad \phi_f(s_t) = Q_t(p) \quad \text{for } p = 3, 4 \tag{3.45}$$

2. Total of playable space remaining.

$$\phi_f(s_t) = (99 - Q_t(1)) + (99 - Q_t(2)) + Q_t(3) + Q_t(4). \tag{3.46}$$

3. Total amount of cards remaining in play.:

$$\phi_f(s_t) = \sum_{c=2}^{99} \Big(K_t(c) + H_t(c) \Big). \tag{3.47}$$

4. Total amount of cards in hand:

$$\phi_f(s_t) = \sum_{c=2}^{99} H_t(c). \tag{3.48}$$

5. Absolute difference in piles that have the same direction, thus two features.

$$\phi_f(s_t) = |Q_t(1) - Q_t(2)| \qquad \phi_f(s_t) = |Q_t(3) - Q_t(4)|. \tag{3.49}$$

6. Sum of numbers in hand:

$$\phi_f(s_t) = \sum_{c=2}^{99} \Big(c \times H_t(c) \Big). \tag{3.50}$$

7. Amount of jumping back pairs pairs still present.

8. Smallest valid play using cards from hand on pile, summed up over all 4 piles.

$$\sum_{p=1}^{2} \min_{c2[2,99], cH_t(c) \quad Q_t(p)>0} |Q_t(p) - c \times H_t(c)| + \sum_{p=3}^{4} \min_{c2[2,99], cH_t(c) \quad Q_t(p)<0} |Q_t(p) - c \times H_t(c)| \tag{3.51}$$

Observe that every feature $f \in F$ of the list above satisfies $\phi_f(s) \geq 0$.

Chapter 4

Optimal Stopping Problem

Optimal stopping problems focus on when to take a certain action to maximise the reward or minimise the cost. For example, consider house selling, in this case you want to sell the house at the moment your reward is maximised. The pas and current state of the housing market is known, whereas the future remains uncertain, creating a problem. The decision whether to sell only depends on what is known. Another example is options trading, which has a similar problem, except instead of a house, we are selling or buying options. Options usually have constraints on when the option can be sold or bought. Optimal stopping problems focus on the optimal decision rule.
In this thesis, the optimal stopping problem is used to try and find an approximate guarantee for ADP, which is done in Chapter 5. In this chapter, the optimal stopping problem is introduced together with one approximation method. The approximation method used is taken from an article by Y. Chen & D. Goldberg, see [10].
Section 4.1 gives a definition of optimal stopping problems. Several required definitions and other preliminaries are stated in section 4.2. The theorem that forms the basis of the approximation method is introduced in section 4.3. Section 4.4 states three di□erent approximate guarantees. Finally, in section 4.5, an algorithm to approximate the optimal stopping problem is given, including confidence interval, computational time and randomness calls.

4.1 Definition optimal stopping problem

Recall the definition of a stochastic process (Definition 1). In addition, recall that $g_t(Y_{[t]})$ is a cost function dependent on the entire process up till t, denoted by $Y_{[t]} = \{Y_1, Y_2, \ldots, Y_n\}$. The definition of a stopping time, as given by [27], can be seen in Definition 4.

Definition 4. *(Stopping Time) The positive integer-valued, possibly infinite, random variable τ is said to be a random time for process $\{Y_n, n \geq 1\}$ if the event $\{\tau = n\}$ is determined by the random variables $Y_1, \ldots, Y_n$. That is, knowing $Y_1, \ldots, Y_n$ tells us whether or not $\tau = n$. If $\mathsf{P}(\tau < \infty) = 1$, then the random time τ is said to be a stopping time. Denote $\mathcal{T}$ as the set containing all stopping time τ.*

As previously mentioned, optimal stopping problems focus on finding the optimal stopping time to minimise the costs. Therefore, the goal is to compute

$$OPT = \inf_{\tau 2T} \mathsf{E}[g_\tau(Y_{[\tau]})]. \tag{4.1}$$

So, an optimal stopping problem consists of a stochastic process and a decision, based on the past and present, to stop or to continue. The decision rule states in what situations one should continue and when one should stop. Note that an optimal stopping problem is not necessarily an MDP, as an optimal stopping problem can be history dependent and hence does not have the Markov property, see equation (3.1).

4.2 Preliminaries

Recall that $\mathbf{Y} = \{Y_t, t \in [1,T]\}$ is a stochastic process and let $g_t(Y_{[t]})$ be a cost function. Define $Z_t = g_t(Y_{[t]})$ and let $\tau \in \mathcal{T}$ be a stopping time in $[1,T]$. This means $\{Z_t, t \geq 1\}$ is also a stochastic process. The goal is to compute

$$OPT = \inf_{\tau 2T} \mathsf{E}[Z_\tau]. \tag{4.2}$$

This means we have an optimal stopping problem. Any relevant information concerning the stochastic process $\mathbf{Y}$ up to time t is denoted by $\mathcal{F}_t$. The set $\mathcal{F}_t$ is called the natural filtration. The definition of a martingale is stated in Definition 5 [26].

Definition 5. *(Martingale) A stochastic process $\{Y_n, n \geq 1\}$ is said to be a martingale process if*

$$\mathsf{E}[|Y_t|] < \infty \quad \textit{for all } t \tag{4.3}$$

and

$$\mathsf{E}[Y_{t+1}|Y_1, Y_2, \ldots, Y_t] = Y_t. \tag{4.4}$$

A consequence of Definition 5 is that, for a martingale $\{Y_t, t \geq 1\}$, the following must hold

$$\mathsf{E}[Y_t] = \mathsf{E}[Y_{t\ \ 1}] = \cdots = \mathsf{E}[Y_1]. \tag{4.5}$$

Next, define a Doob martingale [26] in Proposition 1, which is a stochastic process that by construction is always a martingale.

Proposition 1. *Let $\mathcal{X}, Y_1, Y_2, \ldots$ be arbitrary random variables such that $\mathsf{E}[|\mathcal{X}|] < \infty$, and let $\mathcal{M}_n = \mathsf{E}[\mathcal{X}|Y_1, Y_2, \ldots, Y_t]$. Then $\{\mathcal{M}_t, t \geq 1\}$ is a martingale, also called a Doob martingale.*

Proof. To prove that $\{\mathcal{M}_t, t \geq 1\}$ is a martingale, it needs to satisfy equations (4.3) and (4.4). Equation (4.3) holds since $\mathsf{E}[|\mathcal{X}|] < \infty$. Next we show that equation (4.4) is also satisfied. First the definition of M_n is filled in to obtain

$$\mathsf{E}[\mathcal{M}_{t+1}|Y_1, \ldots, Y_t] = \mathsf{E}[\mathsf{E}[\mathcal{X}, Y_1, \ldots, Y_{t+1}]|Y_1, \ldots, Y_t]. \tag{4.6}$$

Since $\mathsf{E}[\mathcal{X}|U] = \mathsf{E}[\mathsf{E}[\mathcal{X}|Y,U]|U]$ ([26]), where U is an arbitrary random variable, it follows from equation (4.6) that

$$\begin{aligned}\mathsf{E}[\mathcal{M}_{t+1}|Y_1,\ldots,Y_t] &= \mathsf{E}[\mathcal{X}|Y_1,\ldots,Y_t] \\ &= \mathcal{M}_t. \end{aligned} \tag{4.7}$$

This completes the proof of Proposition 1. □

Theorem 5 states the optimal stopping theorem. A proof is available in [26]. Recall that τ is the stopping time introduced in Definition 4.

Theorem 5. *(Optional stopping theorem) Let $\{\mathcal{M}_t, t \geq 1\}$ be a martingale. If either*

1. *$\overline{\mathcal{M}}_t$ are uniformly bounded, where*

$$\overline{\mathcal{M}}_t = \begin{cases} \mathcal{M}_t & \text{if } t \leq T \\ \mathcal{M}_T & \text{if } t \geq T, or, \end{cases} \tag{4.8}$$

2. *τ is bounded, or,*

3. *$\mathsf{E}[\tau] < \infty$, and there is an $L < \infty$ such that*

$$\mathsf{E}[|\mathcal{M}_{t+1} - \mathcal{M}_t||\mathcal{M}_1,\ldots,\mathcal{M}_t] < L, \tag{4.9}$$

then $\mathsf{E}[\mathcal{M}_\tau] = \mathsf{E}[\mathcal{M}_1]$.

4.3 Main theorem

Define $\mathcal{M}_t = \mathsf{E}[\min_{i2[1,T]} Z_i|\mathcal{F}_t]$. By Proposition 1 $\{\mathcal{M}_t, t \geq 1\}$ is a martingale. The following lemma gives us the first step to proofthe main theorem.

Lemma 1.

$$\inf_{\tau 2 T} \mathsf{E}[Z_\tau] = \mathsf{E}[\min_{t2[1,T]} Z_t] + \inf_{\tau 2 T} \mathsf{E}\Big[Z_\tau - \mathsf{E}[\min_{t2[1,T]} Z_t|\mathcal{F}_\tau]\Big], \tag{4.10}$$

Proof. Condition 2 of the optional stopping theorem (Theorem 5) is satisfied, which implies $\mathsf{E}[\mathcal{M}_t] = \mathsf{E}[\mathcal{M}_\tau]$ for all $t \in [1,T]$ and stopping times $\tau \leq T$. Since $\mathsf{E}[\mathcal{X}|U] = \mathsf{E}[\mathsf{E}[\mathcal{X}|Y,U]|U]$, we obtain $\mathsf{E}[\mathcal{M}_1] = \mathsf{E}[\min_{t2[1,T]} Z_t]$. In view of $\mathsf{E}[\mathsf{E}[\mathcal{X}]] = \mathsf{E}[\mathcal{X}]$, for arbitrary random variable $\mathcal{X}$, we have $\mathsf{E}[\mathcal{M}_t] = \mathcal{M}_t$ and therefore

$$\mathsf{E}[\min_{t2[1,T]} Z_t] = \mathsf{E}[\min_{t2[1,T]} Z_t|\mathcal{F}_i] = \mathsf{E}[\min_{t2[1,T]} Z_t|\mathcal{F}_\tau] \quad \forall i \in [1,T], \tau \in \mathcal{T}. \tag{4.11}$$

This implies

$$\inf_{\tau 2 T} \mathsf{E}[Z_\tau] = \inf_{\tau 2 T} \mathsf{E}[Z_\tau] + \mathsf{E}[\min_{t2[1,T]} Z_t] - \mathsf{E}[\min_{t2[1,T]} Z_t|\mathcal{F}_i] \quad \forall i \in [1,T]. \tag{4.12}$$

Since $\mathsf{E}[\min_{t \in [1,T]} Z_t|\mathcal{F}_i]$ is independent of τ, equation (4.12) yields

$$\inf_{\tau \in \mathcal{T}} \mathsf{E}[Z_\tau] = \mathsf{E}[\min_{t \in [1,T]} Z_t] + \inf_{\tau \in \mathcal{T}} \mathsf{E}\Big[Z_\tau - \mathsf{E}[\min_{t \in [1,T]} Z_t|\mathcal{F}_i]\Big] \quad \forall i \in [1,T]. \tag{4.13}$$

Since (4.11) holds for all stopping times, we obtain

$$\inf_{\tau \in \mathcal{T}} \mathsf{E}[Z_\tau] = \mathsf{E}[\min_{t \in [1,T]} Z_t] + \inf_{\tau \in \mathcal{T}} \mathsf{E}\Big[Z_\tau - \mathsf{E}[\min_{t \in [1,T]} Z_t|\mathcal{F}_\tau]\Big]. \tag{4.14}$$

This completes the proof of Lemma 1.

□

Definition 6. *(Z_t^k) Define $Z_t^1 = Z_t$ and*

$$Z_t^{k+1} = Z_t^k - \mathsf{E}[\min_{i \in [1,T]} Z_i^k|\mathcal{F}_t] \quad \forall k \geq 1. \tag{4.15}$$

Using Definition 6 and Lemma 1, gives the initial step for the main theorem.

Lemma 2. *For all $K \geq 1$, the following equality holds*

$$\inf_{\tau \in \mathcal{T}} \mathsf{E}[Z_\tau] = \sum_{k=1}^{K} \mathsf{E}[\min_{t \in [1,T]} Z_t^k] + \inf_{\tau \in \mathcal{T}} \mathsf{E}[Z_\tau^{K+1}]. \tag{4.16}$$

Proof. Proof of this lemma is by induction. First consider $K = 1$. By the definition of Z_t^2, equation (4.10) of Lemma 1 yields

$$\inf_{\tau \in \mathcal{T}} \mathsf{E}[Z_\tau] = \mathsf{E}[\min_{t \in [1,T]} Z_t^1] + \inf_{\tau \in \mathcal{T}} \mathsf{E}[Z_\tau^2]. \tag{4.17}$$

Now assume equation (4.16) holds for some $K \geq 2$. The goal is to show, for $K \geq 2$,

$$\inf_{\tau \in \mathcal{T}} \mathsf{E}[Z_\tau] = \sum_{k=1}^{K+1} \mathsf{E}[\min_{t \in [1,T]} Z_t^k] + \inf_{\tau \in \mathcal{T}} \mathsf{E}[Z_\tau^{K+2}]. \tag{4.18}$$

First, rewrite the term $\sum_{k=1}^{K+1} \mathsf{E}[\min_{t \in [1,T]} Z_t^k]$ to

$$\sum_{k=1}^{K+1} \mathsf{E}[\min_{t \in [1,T]} Z_t^k] = \sum_{k=1}^{K} \mathsf{E}[\min_{t \in [1,T]} Z_t^k] + \mathsf{E}[\min_{t \in [1,T]} Z_t^{K+1}]. \tag{4.19}$$

Second, by Definition 6, $Z_t^{K+2} = Z_t^{K+1} - \mathsf{E}[\min_{i \in [1,T]} Z_i^{K+1}|\mathcal{F}_t]$. This yields

$$\inf_{\tau \in \mathcal{T}} \mathsf{E}[Z_\tau^{K+2}] = \inf_{\tau \in \mathcal{T}} \mathsf{E}\Big[Z_\tau^{K+1} - \mathsf{E}[\min_{t \in [1,T]} Z_t^{K+1}|\mathcal{F}_\tau]\Big]. \tag{4.20}$$

By Proposition 1, $\mathsf{E}[\min_{t \in [1,T]} Z_t^{K+1}|\mathcal{F}_i]$ is a martingale, since it is a Doob martingale.
Since τ is finite, the Optional Stopping Theorem can be applied, implying that $\mathsf{E}[\min_{t \in [1,T]} Z_t^{K+1}] = \mathsf{E}[\min_{t \in [1,T]} Z_t^{K+1}|\mathcal{F}_\tau]$ for all stopping time $\tau \in \mathcal{T}$. Rewrite equation (4.20) as

$$\inf_{\tau \in \mathcal{T}} \mathsf{E}[Z_\tau^{K+2}] = \inf_{\tau \in \mathcal{T}} \mathsf{E}[Z_\tau^{K+1}] - \mathsf{E}[\min_{t \in [1,T]} Z_t^{K+1}]. \tag{4.21}$$

Substituting (4.19) and (4.21) into (4.18), yields (4.16) for $K+1$. This completes the proof of Lemma 1. □

Using Lemma 2, OPT can be rewritten to a sum with K terms plus some remainder term. If K goes to infinity, then a sum with an infinite amount of terms is obtained. The remainder term will go to 0, which is shown in Lemma 3. The definition of almost sure convergence is given [14].

Definition 7. *(Almost sure convergence) We say $\mathcal{Z}_n$ converges almost surely to $\mathcal{Z}$ if*

$$\mathsf{P}(\lim_{n \to \infty} \mathcal{Z}_n = \mathcal{Z}) = 1. \tag{4.22}$$

Lemma 3 shows that the remainder goes to zero, as well as two properties of Z_t^k.

Lemma 3.

$$\lim_{k \to \infty} \inf_{\tau \in T} \mathsf{E}[Z_\tau^k] = 0. \tag{4.23}$$

Additionally Z_t^k is non-negative and a monotone decreasing sequence of random variables in k.

Proof. First, observe that $Z_t^1 = Z_t \geq 0$ and $Z_t^{k+1} = Z_t^k - \mathsf{E}[\min_{i \in [1,T]} Z_i^k | \mathcal{F}_t]$ by Definition 6. Since $Z_t^k \geq \mathsf{E}[\min_{i \in [1,T]} Z_i^k | \mathcal{F}_t]$ because of the minimisation function, $Z_t^k \geq 0$ for all $k \geq 1$. Therefore Z_t^k is non-negative.
Second is to show that $\{Z_t^k, k \geq 1\}$ is a monotone decreasing sequence of random variables, thus $Z_t^{k+1} \leq Z_t^k$ for all $k \geq 1$. Observe that Z_t^k is non-negative and as a consequence $\mathsf{E}[\min_{i \in [1,T]} Z_i^k | \mathcal{F}_t] \geq 0$. This yields

$$Z_t^{k+1} = Z_t^k - \mathsf{E}[\min_{i \in [1,T]} Z_i^k | \mathcal{F}_t] \leq Z_t^k. \tag{4.24}$$

Consequently, $\{Z_t^k, k \geq 1\}$ is a monotone decreasing sequence of random variables.
Next, we proof equation (4.23). Since $Z_t^k \geq 0$ and by the monotone decreasing property of Z_t^k, the sequence $\{Z_t^k, k \geq 1\}$ converges almost surely because of the Monotone Convergence Theorem [35]. The limit of this sequence is unknown. Let $t = T$, then the sequence $\{Z_T^k, k \geq 1\}$ converges almost surely to some random value.
Because $\{Z_T^k, k \geq 1\}$ converges, it also has the Cauchy property [35]. This implies $\{Z_T^{k+1} - Z_T^k, k \geq 1\}$ converges almost surely to 0. Also observe that $\mathsf{E}[\min_{i \in [1,T]} Z_i^k | \mathcal{F}_T] = \min_{i \in [1,T]} Z_i^k]$, since $\mathcal{F}_T$ contains all information about stochastic process until the horizon T. Therefore, by definition of Z_T^{k+1},

$$\min_{t \in [1,T]} Z_i^k = Z_T^k - Z_T^{k+1}, \quad \text{for } k \geq 1. \tag{4.25}$$

It was already shown that the right side of equation (4.25) converges almost surely to 0. Therefore $\{\min_{t \in [1,T]} Z_t^k, k \geq 1\}$ converges almost surely to 0.
Hence for any $j \geq 1$, there exists K_j s.t. $k \geq K_j$ implies

$$\mathsf{P}(\min_{i \in [1,T]} Z_i^k \geq \frac{1}{j}) < \frac{1}{j^2}. \tag{4.26}$$

This statement holds because of the monotonic decreasing property of $\min_{i \in [1,T]} Z_i^k$ and almost sure convergence to 0 implying that for any $d \geq 0$ there is a K such that $\min_{i \in [1,T]} Z_i^K \leq d$.

Let $d = \frac{1}{j}$ where $j \geq 1$, then $\mathsf{P}(\min_{i2[1,T]} Z_i^K \geq \frac{1}{j}) = 0 < \frac{1}{j^2}$ for K su□ciently large. Therefore there exists a strictly increasing sequence of integers $\{K_j^0, j \geq 1\}$ s.t. $\mathsf{P}(\min_{i2[1,T]} Z_i^{K_j^0} \geq \frac{1}{j}) < \frac{1}{j^2}$.

Consider the stopping time τ_j^0 that stops when $Z_t^{K_j^0} \leq \frac{1}{j}$ or at T otherwise. Let I_j^0 be the indicator function for the event $\{\min_{i2[1,T]} Z_i^{K_j^0} > \frac{1}{j}\}$. This creates the following inequality

$$Z_{\tau_j^0}^{K_j^0} \leq \frac{1}{j} + I_j^0 Z_T^{K_j^0}. \tag{4.27}$$

Because Z_T^k is monotone decreasing, it follows that $\{Z_T^{K_j^0}, j \geq 1\}$ is also monotone decreasing because it is a subset.

Next,consider the sequence $\{I_j^0\}$ and apply the Borel-Cantelli Lemma[1]. Using equation (4.26) shows that the following holds: $\sum_{j=1}^{1} \mathsf{P}(I_j^0) \leq \sum_{j=1}^{1} \frac{1}{j^2} = \frac{\pi^2}{6} < \infty$. This implies that $\mathsf{P}(\limsup_{j\,!\,1} I_j^0) = 0$, therefore $I_j^0 = 0$ after some finite time. Therefore $\frac{1}{j} + I_j^0 Z_T^{K_j^0}$ converges almost surely to 0. By inequality (4.27) and the convergence of the right side of the inequality, it follows $\lim_{j\,!\,1} \mathsf{E}[Z_{\tau_j^0}^{K_j^0}] = 0$. Therefore

$$\lim_{j\,!\,1} \inf_{\tau 2 T} \mathsf{E}[Z_\tau^{K_j^0}] = 0, \tag{4.28}$$

because of non-negativity and the almost sure convergence of a stopping time to 0.

This proves the convergence of a subsequence of $\{\inf_{\tau 2 T} \mathsf{E}[Z_\tau^j], j \geq 1\}$ to 0. Let p_1 and p_2 be two consecutive elements of the subsequence. Adding an element q where $p_1 \leq q \leq p_2$ still implies convergence. Because of the monotone decreasing property, any elements of $\{\inf_{\tau 2 T} \mathsf{E}[Z_\tau^j], j \geq 1\}$ not in the subsequence can be added without altering the convergence property. Hence the sequence $\{\inf_{\tau 2 T} \mathsf{E}[Z_\tau^j], j \geq 1\}$ also converges to 0. Let $k \to \infty$, then $\lim_{k\,!\,1} \inf_{\tau 2 T} \mathsf{E}[Z_\tau^k] = 0$. This completes the proof of Lemma 3. □

Now follows the main results, which is a theorem that states that OPT can be rewritten to a sum with an infinite amount of terms.

Theorem 6.

$$OPT = \inf_{\tau 2 T} \mathsf{E}[Z_\tau] = \sum_{k=1}^{1} \mathsf{E}[\min_{t2[1,T]} Z_t^k]. \tag{4.29}$$

Proof. Let K to infinity in Lemma 2 and apply Lemma 3 yields the result. □

Define

$$E_K = \sum_{k=1}^{K} \mathsf{E}[\min_{t2[1,T]} Z_t^k]. \tag{4.30}$$

The remainder of the chapter will consider E_K, how to approximate it, and the quality of the approximation compared to OPT.

[1] The Borel-Cantelli Lemma states, given some sequence of events fE_ng, that if $\sum_{n=1}^{1} \mathsf{P}(E_n) < 1$, then $\mathsf{P}(\limsup_{n\,!\,1} E_n) = 0$.[26]

4.4 Approximate guarantees and rate of convergence

This section states three different approximate guarantees; two assume normalization of Z_t and one assumes non-negativity of Z_t.

4.4.1 Normalized

This section starts by stating and proving Lemma 4, because proving the main results requires Lemma 4.

Lemma 4. *Suppose with probability 1 $Z_t \in [0, U]$ for all $t \in [1, T]$. Then for all $k \geq 1$, w.p.1 $\min_{t \in [1,T]} Z_t^k \leq \frac{U}{k}$.*

Proof. Rewriting Definition 6 yields $Z_T^{k+1} = Z_T - \sum_{i=1}^{k} \min_{t \in [1,T]} Z_t^i$. By Lemma 3, Z_T^{k+1} is non-negative and as a consequence

$$Z_T \geq \sum_{i=1}^{k} \min_{t \in [1,T]} Z_t^i. \tag{4.31}$$

Because of the monotone decreasing property of $\min_{t \in [1,T]} Z_t^i$ (see Lemma 3), it follows that $\sum_{i=1}^{k} \min_{t \in [1,T]} Z_t^i \geq k \times \min_{t \in [1,T]} Z_t^i$. Recall that $Z_t \in [0, U]$ and therefore

$$U \geq Z_t \geq k \times \min_{t \in [1,T]} Z_t^i \quad \forall k \geq 1. \tag{4.32}$$

Dividing the above inequality by k gives us the statement of Lemma 4. This completes the proof. □

Note that Lemma 4 does not assume normalization of Z_t. Theorem 7 states an approximate guarantee for E_K.

Theorem 7. *Suppose w.p.1 $Z_t \in [0, 1]$ for all $t \in [1, T]$. Then for all $k \geq 1$,*

$$0 \leq OPT - E_k \leq \frac{1}{k+1}. \tag{4.33}$$

Proof. Since Z_k is normalised, fix $U = 1$. Define τ_{k+1} as the stopping time that stops when $Z_t^{k+1} \leq \frac{1}{k+1}$ or stop at T otherwise. By Lemma 4, $\min_{t \in [1,T]} Z_t^{k+1} \leq \frac{1}{k+1}$. This implies that the condition $Z_t^{k+1} \leq \frac{1}{k+1}$ of stopping time τ_{k+1} will be satisfied for some $t \in [1, T]$, thus $\mathsf{E}[Z_{\tau_{k+1}}^{k+1}] = \mathsf{E}[\min_{t \in [1,T]} Z_t^{k+1}]$. If $Z_t^{k+1} \leq \frac{1}{k+1}$, then $\mathsf{E}[Z_t^{k+1}] \leq \frac{1}{k+1}$ as well. Combining the above gives

$$\mathsf{E}[Z_{\tau_{k+1}}^{k+1}] = \mathsf{E}[\min_{t \in [1,T]} Z_t^{k+1}] \leq \frac{1}{k+1}. \tag{4.34}$$

Also $\inf_{\tau \in T} \mathsf{E}[Z_\tau^{k+1}] \leq \mathsf{E}[Z_{\tau_{k+1}}^{k+1}]$. Lemma 2 states $OPT = \sum_{i=1}^{k} \mathsf{E}[Z_t^i] + \inf_{\tau \in T} \mathsf{E}[Z_\tau^{k+1}]$. Combining the above implies

$$OPT - \sum_{i=1}^{k} \mathsf{E}[Z_t^i] = \inf_{\tau \in T} \mathsf{E}[Z_\tau^{k+1}] \leq \frac{1}{k+1}. \tag{4.35}$$

This completes the proof of Theorem 7. □

The next lemma is required to show that the bound of Theorem 7 cannot be improved.

Lemma 5. *Consider the setting for $n \geq 1$ where $T = 2$, $D = 1$, $Z_t = Y_t$ for $t \in [1,2]$ and $\mathsf{P}(Y_1 = \frac{1}{n}) = 1$, $\mathsf{P}(Y_2 = 1) = \frac{1}{n}$, $\mathsf{P}(Y_2 = 0) = 1 - \frac{1}{n}$. In this setting, for all $k \geq 1$, $OPT - E_k = \frac{1}{n} \times (1 - \frac{1}{n})^k$.*

Proof. First, show by induction that $\mathrm{Var}[Z_1^k] = 0$ for all $k \geq 1$. Consider $k = 1$, then $\mathrm{Var}[Z_1^1] = 0$ because $Z_1 = \frac{1}{n}$ and therefore constant. Next assume that $\mathrm{Var}[Z_1^k] = 0$ for some $k \geq 1$. Then it must also hold for $\mathrm{Var}[Z_1^{k+1}]$, which by Definition 6 is equal to $\mathrm{Var}[Z_1^k - \mathsf{E}[\min_{i2[1,2]} Z_1^k | \mathcal{F}_1]]$. Observe for arbitrary random variable X that $\mathrm{Var}[\mathsf{E}[X]] = 0$ since $\mathsf{E}[X]$ is a number. Combined with the induction hypothesis, it follows that $\mathrm{Var}[Z_1^{k+1}] = 0$. This completes the induction and hence $\mathrm{Var}[Z_1^k] = 0$ for all $k \geq 1$.
Recall Definition 5 of a Martingale and the fact that $\{Z_t^k, t \in [1,T]\}$ is a martingale by Proposition 1. Because of $\mathsf{E}[Z_1] = \frac{1}{n}$ and $\mathsf{E}[Z_2] = \frac{1}{n}$, the fact that $\{Z_t^k, t \in [1,T]\}$ and by the Optional Stopping Theorem 5, obtain $OPT = \frac{1}{n}$.
Additionally, by the Optional Stopping Theorem $\inf_{\tau 2 T} \mathsf{E}[Z_\tau^{k+1}] = \mathsf{E}[Z_1^{k+1}]$ for all $k \geq 1$, combined with Lemma 2, it is su□cient to show that $\mathsf{E}[Z_1^k] = \frac{1}{n} \times (1 - \frac{1}{n})^{k\ 1}$. Since $\mathrm{Var}[Z_1^k] = 0$, the last step is to proof $\mathsf{P}(Z_1^k = \frac{1}{n} \times (1 - \frac{1}{n})^{k\ 1}) = 1$ for all $k \geq 1$. This is done by induction. First consider $k = 1$, thus $\mathsf{P}(Z_1 = \frac{1}{n}) = 1$, which holds by definition of Z_1. Second, assume $\mathsf{P}(Z_1^k = \frac{1}{n} \times (1 - \frac{1}{n})^{k\ 1}) = 1$ for some $k \geq 1$. The martingale property and induction hypotheses imply

$$Z_1^k = \frac{1}{n} \times (1 - \frac{1}{n})^{k\ 1}, \text{ and } Z_2^k = \begin{cases} (1 - \frac{1}{n})^{k\ 1} & \text{w.p. } \frac{1}{n} \\ 0 & \text{w.p. } 1 - \frac{1}{n}. \end{cases} \tag{4.36}$$

Therefore

$$Z_1^{k+1} = Z_1^k - \mathsf{E}[\min_{t2[1,2]} Z_t^k] = \frac{1}{n}(1 - \frac{1}{n})^{k\ 1} - (\frac{1}{n})^2(1 - \frac{1}{n})^{k\ 1} = \frac{1}{n}(1 - \frac{1}{n})^k, \tag{4.37}$$

completing the proof of Lemma 5. □

Theorem 8 shows that the linear convergence of Theorem 7 cannot be improved.

Theorem 8. *For any $n \geq 2$, there exists an optimal stopping problem with $T = 2$, $\mathsf{P}(Z_t \in [0,1]) = 1$ for $t \in [1,2]$, yet $OPT - E_k \geq \frac{1}{4n}$ for all $k \leq n$.*

Proof. Note that for any $n \geq 2$, $(1 - \frac{1}{n})^k \geq \frac{1}{4}$ for all $k \leq n$. Lemma 5 states that the optimal stopping problem constructed yields $OPT - E_k \geq \frac{1}{4n}$. This completes the proof of Theorem 8. □

4.4.2 Normalized and prophet equations

This section states an approximate guarantee, that relies on the prophet inequalities [11]. Let $z \in [0,1]$, $h_1(z) = (1 - z)\log(\frac{1}{1\ z})$ and for $k \geq 2$, $h_k(z) = h_1(h_{k\ 1}(z))$. The functions h_k are the prophet equations. Lemma 6 is required for the proof in the upcoming theorem, and [11] proofs Lemma 6.

Lemma 6. *Suppose* $\mathsf{P}(Z_t \in [0,1]) = 1$ *for all* $t \in [1,T]$. *Then*

$$OPT - \mathsf{E}[\min_{t \in [1,T]} Z_t] \leq h_1(OPT). \tag{4.38}$$

Theorem 9. *Suppose w.p.1* $Z_t \in [0,1]$ *for all* $t \in [1,T]$. *Then for all* $k \geq 1$,

$$0 \leq OPT - E_k \leq h_k(OPT). \tag{4.39}$$

In addition, for each fixed $z \in [0,1]$, $\{h_k(z), k \geq 1\}$ *is a monotone decreasing sequence converging to 0; and* $\lim_{z \# 0} h_1(z) = \lim_{z \uparrow 1} h_1(z) = 0$.

Proof. First assume several basic properties of $h_1(z)$. The properties are easily verified by plotting the function. First $h_1(z) \in [0,1]$ for all $z \in [0,1]$. Second $h_1(z) \leq z$ for all $z \in [0,1]$ from which follows that each $\{h_k(z), k \geq 1\}$ decreases monotone (by induction argument and definition of $h_k(z)$). Third h_1 strictly increases on $[0, 1 - e^{-1}]$. Fourth $h_1(z) \leq e^{-1}$ for all $z \in [0,1]$.

Next, show that $OPT - E_k \leq h_k(OPT)$, which by Lemma 2 is equivalent to

$$\inf_{\tau \in T} \mathsf{E}[Z_\tau^{k+1}] \leq h_k(OPT). \tag{4.40}$$

Proof inequality (4.40) by induction. First, consider $k = 1$. Lemma 2 gives $\inf_{\tau \in T} \mathsf{E}[Z_\tau^{k+1}] = OPT - E_k$. Since $Z_t \in [0,1]$, Lemma 6 can be applied, which completes the first part of the induction.

Second assume inequality (4.40) holds for some $k \geq 1$. Definition 6 and Lemma 6 yields

$$\inf_{\tau \in T} \mathsf{E}[Z_\tau^{k+2}] = \inf_{\tau \in T} \mathsf{E}[Z_\tau^{k+1}] - \mathsf{E}[\min_{t \in [1,T]} Z_t^{k+1}] \leq h_1\Big(\inf_{\tau \in T} \mathsf{E}[Z_\tau^{k+1}]\Big). \tag{4.41}$$

The induction hypotheses states $\inf_{\tau \in T} \mathsf{E}[Z_\tau^{k+1}] \leq h_k(OPT)$. Recall $h_k(OPT) \leq h_1(OPT) \leq e^{-1} \leq 1 - e^{-1}$, and $h_1(z)$ is increasing on $z \in [0, 1 - e^{-1}]$. Therefore

$$h_1\Big(\inf_{\tau \in T} \mathsf{E}[Z_\tau^{k+1}]\Big) \leq h_1\Big(h_k(OPT)\Big) = h_{k+1}(OPT). \tag{4.42}$$

This implies that $\inf_{\tau \in T} \mathsf{E}[Z_\tau^{k+1}] \leq h_k(OPT)$ for all $k \geq 1$. This completes the proof of Theorem 9. □

4.4.3 Non-normalized

Theorems 7 and 9 state guarantees for instances where $Z_t \in [0,1]$. This section states an approximate guarantee for non-normalised instances. This implies a lower rate of convergence.

Theorem 10. *Assume* $\{Z_t, t \geq 1\}$ *is non-negative, then for all* $k \geq 1$,

$$OPT - E_k \leq 2 \times \Big(\frac{\mathsf{E}[(Z_T)^2]}{OPT^2}\Big)^{\frac{1}{3}} \times k^{-\frac{1}{3}} \times OPT \tag{4.43}$$

Proof. By Lemma 2 and the non-negative property of $\mathsf{E}[Z_t^k]$, it follows that $OPT \geq \sum_{i=1}^k \mathsf{E}[Z_t^i]$. Also $\sum_{i=1}^k \mathsf{E}[Z_t^i] \geq k \times \mathsf{E}[\min_{t \in [1,T]} Z_t^k]$, since the sum is larger then k times the minimum. This implies for all $k \geq 1$,

$$\mathsf{E}[\min_{t \in [1,T]} Z_t^k] \leq \frac{1}{k} \times OPT. \tag{4.44}$$

Using Lemma 2 and rewriting inequality (4.43), it is su□cient to show that

$$\inf_{\tau 2 T} \mathsf{E}[Z_\tau^{k+1}] \leq 2 \times \Big(OPT \times \mathsf{E}[(Z_T)^2] \times k^{\ 1} \Big)^{\frac{1}{3}}. \tag{4.45}$$

Let $z_k = \Big(\mathsf{E}[(Z_t)^2] \times OPT \times k^{\ 1} \Big)^{\frac{1}{3}}$. Define the stopping time τ_k as follows: stops the first time $Z_t^{k+1} \leq z_k$ and stops at T otherwise. Let I denote the step function. Then $Z_{\tau_k}^{k+1} \leq z_k + I(\min_{t2[1,T]} Z_t^{k+1} > z_k) Z_T^{k+1}$, combined with $Z_T^{k+1} \leq Z_T$, implies that $Z_{\tau_k}^{k+1} \leq z_k + I(\min_{t2[1,T]} Z_t^{k+1} > z_k) Z_T$. Taking expectation on both sides gives

$$\mathsf{E}[Z_{\tau_k}^{k+1}] \leq z_k + \mathsf{E}\Big[I(\min_{t2[1,T]} Z_t^{k+1} > z_k) Z_T \Big], \tag{4.46}$$

where by definition of z_k, it follows that $\mathsf{E}[z_k] = z_k$. Now apply the Cauchy-Schwarz inequality[2] on $\mathsf{E}\Big[I(\min_{t2[1,T]} Z_t^{k+1} > z_k) Z_T \Big]$ in the above inequality to obtain

$$\mathsf{E}[Z_{\tau_k}^{k+1}] \leq z_k + \Big(\mathsf{E}[(Z_T)^2] \Big)^{\frac{1}{2}} \times \Big(\mathsf{P}(\min_{t2[1,T]} Z_t^{k+1} > z_k) \Big)^{\frac{1}{2}}. \tag{4.47}$$

Next, apply Markov's inequality[3] and inequality (4.44) on $\mathsf{P}(\min_{t2[1,T]} Z_t^{k+1} > z_k)$, which yields $\mathsf{P}(\min_{t2[1,T]} Z_t^{k+1} > z_k) \leq \frac{k^{\ 1}\ OPT}{z_k}$. Therefore, rewriting inequality (4.47) gives

$$\mathsf{E}[Z_{\tau_k}^{k+1}] \leq z_k + \Big(\frac{\frac{1}{k} \times OPT \times \mathsf{E}[(Z_T)^2]}{z_k} \Big)^{\frac{1}{2}} = 2 z_k. \tag{4.48}$$

Filling in the definition of z_k completes the proof of Theorem 10. □

4.5 Algorithm

Theorem 6 states a method to compute the optimal costs for an optimal stopping problem. Instead of computing the optimum, the approximation E_k is computed. Theorems 7, 9 and 10 give bounds on the approximation E_k as well as a rate of convergence. In this section, an algorithm for approximating E_k will be stated, including computational time, calls to randomness and confidence interval.

In section 4.5.1 the preliminaries of the algorithm are given. Section 4.5.2 gives the pseudo-code of the algorithm to approximate E_k. Finally, section 4.5.3 states the relevant results of the algorithm.

4.5.1 Preliminaries

Define $\eta \in \mathcal{N}^t$ as some instance up to time $t \in [1, T]$. Then $Y(\eta)$ is the event conditioned on $\{Y(t) = \eta\}$. Let D be the dimensions of some instance Y_1. Assume that there exists some base simulator $\mathcal{B}$ that has the following properties:

- $\mathcal{B}(0, \emptyset)$ returns an independent sample path Y that is unconditioned,

[2] Cauchy-Schwarz inequality: $|\mathsf{E}[XY]|^2 \quad \mathsf{E}[X^2]\mathsf{E}[Y^2]$

[3] Markov's Inequality: $\mathsf{P}(X \quad a) \quad \frac{\mathbb{E}[X]}{a}$

- $\mathcal{B}$ takes C units of computational time to generate Y, and
- C depends on T, D, but NOT on $t, \eta, \mathcal{B}$.

Next, state some assumptions on the computational and memory costs. First computing $g_t(\eta)$ takes G computational time, where G can depend on T and D, but not on t, η and $\mathcal{B}$. Furthermore addition, subtraction, multiplication, dividing and computing $\max()$ and $\min()$ of two numbers take one computational time. Reading writing and storing input has no cost. Let $\epsilon, \delta \in (0,1)$. Define

$$\overline{N}(\epsilon, \delta) = \lceil \frac{1}{2\epsilon^2} \log(\frac{2}{\delta}) \rceil, \tag{4.49}$$

which is plotted in figure 4.1.

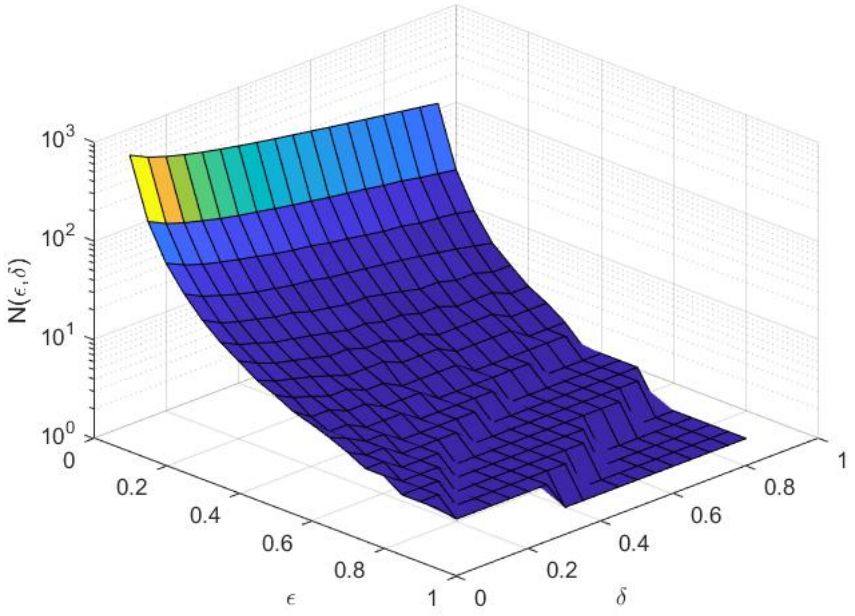

Figure 4.1: Plot of $\overline{N}(\epsilon, \delta)$ for different δ and ϵ.

Define for $k \geq 1$,

$$f_k(\epsilon, \delta) = 10^{2(k-1)^2} \times \epsilon^{-2(k-1)} \times (T+2)^{k-1} \times \Big(1 + \log(\frac{1}{\delta}) + \log(\frac{1}{\epsilon}) + \log(T)\Big)^{k-1}. \tag{4.50}$$

The function $f_k(\epsilon, \delta)$ is plotted for $T = 50$, different k, ϵ and δ in figure 4.2.
Lemma 7 states a recursive relation of $f_k(\epsilon, \delta)$. The proof is omitted and instead we refer the reader to [10].

Lemma 7. *For all* ϵ, $\delta \in (0,1)$ *and* $k \geq 1$,

$$f_{k+1}(\epsilon, \delta) \geq \Big(\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4}) + 1\Big) \times (T+2) \times f_k\Big(\frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})T}\Big). \tag{4.51}$$

The upcoming proofs use Hoeffding's inequality [2], which is stated in Lemma 8.

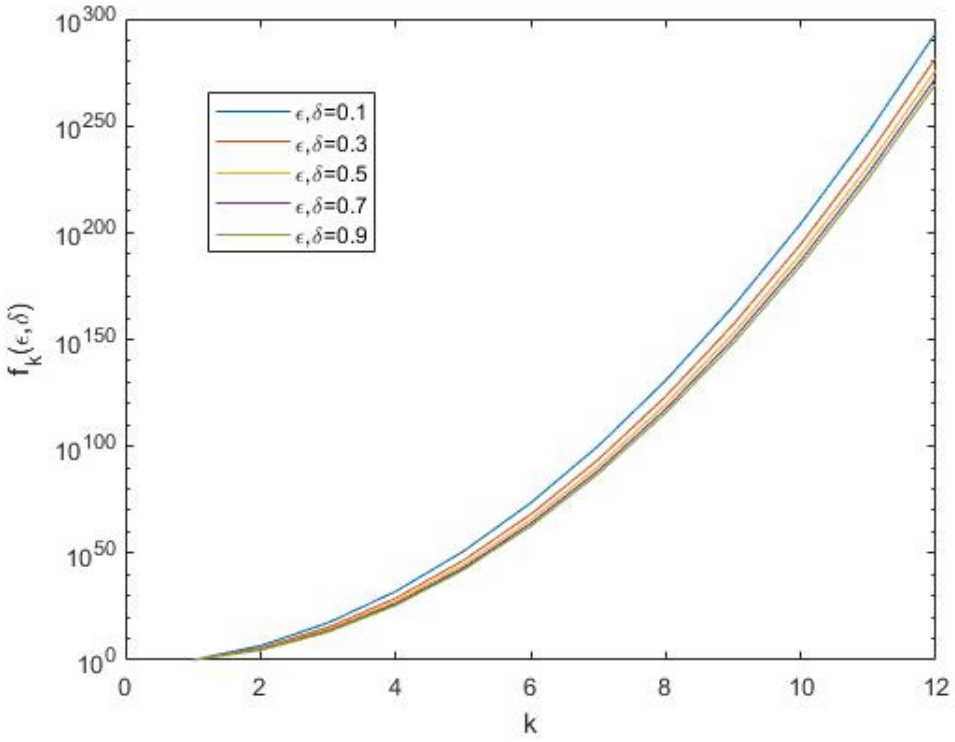

Figure 4.2: Plot of $f_k(\epsilon, \delta)$ for $T = 50$.

Lemma 8. *(Hoe ding's inequality) Suppose that for some* $n \geq 1$ *and* $U > 0$, $\{X_i, i \in [1, n]\}$ *are i.i.d., and* $\mathsf{P}(X_1 \in [0, U]) = 1$. *Then*

$$\mathsf{P}\Big(|n^{\ 1}\sum_{i=1}^{n} X_i - \mathsf{E}[X_1]| \geq \eta\Big) \leq 2\exp\left(-\frac{2\eta^2 n}{U^2}\right). \tag{4.52}$$

4.5.2 Pseudo-code algorithm

Recall that

$$E_K = \sum_{k=1}^{K} D_k = \sum_{k=1}^{K} \mathsf{E}[\min_{t2[1,T]} Z_t^k]. \tag{4.53}$$

First, state the algorithm for approximating Z_t^k given (partial) sample-path η, denoted by $\mathcal{B}^k$, see algorithm 7. Next, algorithm $\hat{\mathcal{B}}^k$, that obtains an ϵ-approximation of D_k. See algorithm 8. Algorithm $\hat{\mathcal{B}}^k$ computes the approximation E_K by computing D_k for all $k \in [1, K]$.

Algorithm 7: Approximate $Z_t^k(\eta)$

Result: ϵ-approximation to $Z_t^k(\eta)$ w.p. at least $1-\delta$
Algorithm $\mathcal{B}^1(t,\eta,\epsilon,\delta)$;
Return $g_t(\eta)$;

Algorithm $\mathcal{B}^{k+1}(t,\eta,\epsilon,\delta)$ (for $k \geq 1$);
Create a length-$\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})$ vector A^0 ;
for *i=1 to* $\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})$ do
    Generate an independent call to $\mathcal{B}(t,\eta)$ and store in D by T matrix A^1 ;
    Create a length-T vector A^2 ;
    for *j=1 to T* do
        Generate an independent call to $\mathcal{B}^k\left(j, A^1_{[j]}, \frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})T}\right)$ and store in A^2_j ;
    end
    Compute the minimum value of A^2 and store in A^0_i ;
end
Generate an independent call to $\mathcal{B}^k(t,\eta,\frac{\epsilon}{2},\frac{\delta}{2})$ and store as variable A^3 ;
Return $A^3 - (\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})} A^0_i$;

Algorithm 8: Approximate D_k

Result: ϵ-approximation to D_k w.p. at least $1-\delta$
Algorithm $\hat{\mathcal{B}}^k(\epsilon,\delta)$;
Create a length-$\overline{N}(\frac{\epsilon}{2},\frac{\delta}{2})$ vector A^0 ;
for *i=1 to* $\overline{N}(\frac{\epsilon}{2},\frac{\delta}{2})$ do
    Generate an independent call to $\mathcal{B}(0,\emptyset)$ and store in D by T matrix A^1 ;
    Create a length-T vector A^2 ;
    for *j=1 to T* do
        Generate an independent call to $\mathcal{B}^k\left(j, A^1_{[j]}, \frac{\epsilon}{2}, \frac{\delta}{2\overline{N}(\frac{\epsilon}{2},\frac{\delta}{2})T}\right)$ and store in A^2_j ;
    end
    Compute the minimum value of A^2 and store in A^0_i ;
end
Return $(\overline{N}(\frac{\epsilon}{2},\frac{\delta}{2}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{2},\frac{\delta}{2})} A^0_i$;

4.5.3 Main algorithmic results

Algorithm 8 allows a trade-o between the accuracy parameters ϵ, δ and the computational costs. Even for the full path dependence and high-dimensionality instances, the algorithm can for any ϵ compute an ϵ-approximation in polynomial time. The proof of the main results of $\hat{\mathcal{B}}^k$ require the relevant results for $\mathcal{B}^k$ in Lemma 9.

Lemma 9. *For all $k \geq 1$, $t \in [1, T]$, $\eta \in \mathcal{N}^t$, $\epsilon, \delta \in (0, 1)$, algorithm $\mathcal{B}^k$ achieves the following when evaluated on $t, \eta, \epsilon, \delta$. In total computational time at most $(C + G + 1) f_k(\epsilon, \delta)$ and with only access to randomness at most $f_k(\epsilon, \delta)$ calls to the base simulator $\mathcal{B}$, returns a random number X satisfying* $\mathsf{P}(|X - Z_t^k(\eta)| \geq \epsilon) \leq \delta$.

Proof. Three elements need to be shown: computational costs, access to randomness and confidence interval. Using induction shows this by considering all three elements in the $k = 1$ and $k \geq 1$ case.
Consider the case where $k = 1$, then $\mathcal{B}^1(t, \eta, \epsilon, \delta)$ will output $Z_t(\eta)$ with no error, computational cost G and no calls to $\mathcal{B}$. Therefore satisfies the confidence interval and calls to randomness. Since $f_1(\epsilon, \delta) \geq 1$, it also satisfies the computational costs.
Next, assume that Lemma 9 holds for some $k \geq 1$, then show that the lemma holds for $k + 1$. First, proof of the confidence interval results. The induction hypothesis states that $\mathcal{B}^k$ outputs a random number X satisfying

$$\mathsf{P}(|X - Z_t^k(\eta)| \geq \epsilon) \leq \delta. \tag{4.54}$$

Let $\{X_i, i \in [1, \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]\}$ be an i.i.d. sequence of random variables each distributed as $\min_{j2[1,T]} Z_j^k(Y(\eta)_{[j]})$. Then $\{X_i, i \in [1, \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]\}$ and $\{A_i^0, i \in [1, \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]\}$ are constructed on the same probability space subject to $|X_i - A_i^0| < \frac{\epsilon}{4}$ for all i with probability at least $1 - \frac{1}{\delta}$.
First, show the common probability claim. Second, show the error and probability bound.
A probability space of a random variable is defined as $\{\omega, F, \mathsf{P}\}$, where ω is the sample space or the space of outcomes, F is the set of events and P is the set of probabilities of events. Two random variables (1 and 2) have a common probability space when $\omega_1 = \omega_2$, $\mathcal{F}_1 = \mathcal{F}_2$ and $\mathsf{P}_1 = \mathsf{P}_2$. Since X_i and A_i both have $i \in [1, \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]$ and both take on values of Z_i^k in the real numbers, hence $\omega_1 = \omega_2$. Since both X_i and A_i^0 depend on η as input, they also share the same events, implying $\mathcal{F}_1 = \mathcal{F}_2$. X_i has a distribution of events which the algorithm $\mathcal{B}$ uses for computing A_i^0, hence $\mathsf{P}_1 = \mathsf{P}_2$.
Now to proof the statement that for all $i \in [1, \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]$

$$\mathsf{P}(|X_i - A_i^0| \geq \frac{\epsilon}{4}) \leq \frac{\delta}{4}. \tag{4.55}$$

First, consider only one $i \in [1, \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]$. The definitions of X_i and A_i^0 are filled in to obtain

$$\mathsf{P}(|X_i - A_i^0| \geq \frac{\epsilon}{4}) = \mathsf{P}\Big(|\min_{j2[1,T]} Z_j^k(Y(\eta)_{[j]}) - \min_{j2[1,T]} \mathcal{B}^k(j, A^1_{[j]}, \frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})T})| \geq \frac{\epsilon}{4}\Big) \tag{4.56}$$

The $\min()$ function satisfies the Lipschitz property[4], with $m = 1$. Therefore, the following equation bounds equation (4.56).

$$\leq \mathsf{P}\Big(\cup_{j2[1,T]} |Z_j^k(Y(\eta)_{[j]})) - \mathcal{B}^k(j, A^1_{[j]}, \frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})T})| \geq \frac{\epsilon}{4}\Big). \tag{4.57}$$

Next, apply Boole's inequality[5] on equation (4.57), which yields

$$\leq \sum_{j=1}^{T} \mathsf{P}\Big(|Z_j^k(Y(\eta)_{[j]})) - \mathcal{B}^k(j, A^1_{[j]}, \frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})T})| \geq \frac{\epsilon}{4}\Big). \tag{4.58}$$

Observe that equation (4.54) bounds every term in the summation of equation (4.58) to obtain

$$\sum_{j=1}^{T} \mathsf{P}\Big(|Z_j^k(Y(\eta)_{[j]})) - \mathcal{B}^k(j, A^1_{[j]}, \frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})T})| \geq \frac{\epsilon}{4}\Big) \leq \sum_{j=1}^{T} \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})T} = \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})}. \tag{4.59}$$

Therefore, for a single $i \in [1, \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]$, it follows that $\mathsf{P}(|X_i - A_i^0| \geq \frac{\epsilon}{4}) \leq \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})}$. Equation (4.55) needs to hold for every $i \in [1, \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]$ and therefore requires $\mathsf{P}(\cap_{i2[1,\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]} |X_i - A_i^0| \geq \frac{\epsilon}{4})$. Since the intersection set is smaller than the union set, and by applying Boole's inequality, yields

$$\mathsf{P}(\cap_{i2[1,\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]} |X_i - A_i^0| \geq \frac{\epsilon}{4}) \leq \mathsf{P}(\cup_{i2[1,\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]} |X_i - A_i^0| \geq \frac{\epsilon}{4}) \tag{4.60}$$

$$\leq \sum_{i=1}^{\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]} \mathsf{P}(|X_i - A_i^0| \geq \frac{\epsilon}{4}). \tag{4.61}$$

Applying inequality (4.59) further bounds equation (4.61) by

$$\mathsf{P}(\cap_{i2[1,\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]} |X_i - A_i^0| \geq \frac{\epsilon}{4}) \leq \sum_{i=1}^{\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]} \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})} = \frac{\delta}{4}. \tag{4.62}$$

Hence, $\mathsf{P}(|X_i - A_i^0| \geq \frac{\epsilon}{4}) \leq \frac{\delta}{4}$ for all $i \in [1, \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]$. This proves that $\{X_i, i \in [1, \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]\}$ and $\{A_i^0, i \in [1, \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})]\}$ are constructed on the same probability space.
Apply Lemma 8 on $\{X_i\}$, with parameters $\eta = \frac{\epsilon}{4}$, $U = 1$ and $n = \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})$. This yields

$$\mathsf{P}\Big(|\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})^{\ 1} \sum_{i=1}^{\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})} X_i - \mathsf{E}[X_1]| \geq \frac{\epsilon}{4}\Big) \leq 2\exp\Big(-2(\frac{\epsilon}{4})^2 \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})\Big). \tag{4.63}$$

Rewriting the right side of the above equation yields

$$\mathsf{P}\Big(|\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})^{\ 1} \sum_{i=1}^{\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})} X_i - \mathsf{E}[X_1]| \geq \frac{\epsilon}{4}\Big) \leq \frac{\delta}{4}. \tag{4.64}$$

[4] Lipschitz property of function l: there exists a m 1 such that $jl(x_1)$ $l(x_2)j$ $mj(x_1$ $x_2)j$
[5] Boole's inequality: $\mathsf{P}(\int_i Y_i)$ $\sum_i \mathsf{P}(Y_i)$

Since $\{X_i\}$ and $\{A_i^0\}$ are constructed on the same probability space, the event $\{|X_i - A_i^0| < \frac{\epsilon}{4}, \forall i\}$ implies

$$\mathsf{P}\Big(|(\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})} A_i^0 - (\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})} X_i| \geq \frac{\epsilon}{4}\Big) \leq \frac{\delta}{4}, \tag{4.65}$$

which means that, if the individual events hold, then the average must also be smaller then $\frac{\epsilon}{4}$. Combining equations (4.64) and (4.65) using Boole's inequality, yields

$$\mathsf{P}\Big(\Big[|(\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})} A_i^0 - (\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})} X_i| \geq \frac{\epsilon}{4}\Big] \cup \tag{4.66}$$

$$\Big[|\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})} X_i - \mathsf{E}[X_1]| \geq \frac{\epsilon}{4}\Big]\Big) \leq \frac{\delta}{2}. \tag{4.67}$$

Next, apply the triangle inequality[6] to inequality (4.67) to obtain

$$\mathsf{P}\Big(|(\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})} A_i^0 - \mathsf{E}[X_1]| \geq \frac{\epsilon}{2}\Big) \leq \frac{\delta}{2}. \tag{4.68}$$

The induction hypothesis states that $\mathsf{P}(|A^3 - Z_t^k(\eta)| \geq \frac{\epsilon}{2}) \leq \frac{\delta}{2}$, since A^3 uses $\mathcal{B}^k$. Applying Definition 6, $Z_t^k(\eta) - \mathsf{E}[X_1] = Z_t^{k+1}(\eta)$, and the triangle inequality to inequality (4.68) yields

$$\delta \geq \mathsf{P}\Big(|\mathsf{E}[X_1] - (\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})} A_i^0| \geq \frac{\epsilon}{2}\Big) + \mathsf{P}(|A^3 - Z_t^k(\eta)| \geq \frac{\epsilon}{2}) \tag{4.69}$$

$$\geq \mathsf{P}\Big(|\mathsf{E}[X_1] - (\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})} A_i^0 + A^3 - Z_t^k(\eta)| \geq \epsilon\Big) \tag{4.70}$$

$$= \mathsf{P}\Big(|A^3 - (\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})} A_i^0 - Z_t^{k+1}(\eta)| \geq \epsilon\Big) \tag{4.71}$$

$$= \mathsf{P}\Big(|\mathcal{B}^{k+1}(t,\eta,\epsilon,\delta) - Z_t^{k+1}(\eta)| \geq \epsilon\Big). \tag{4.72}$$

Hence $\mathcal{B}^{k+1}$ outputs a random number X satisfying $\mathsf{P}(X - Z_t^{k+1}(\eta)| \geq \epsilon) \leq \delta$. This proves the error and probability bound of Lemma 9.

Next, proof the statement about the amount of calls to the base simulator $\mathcal{B}$. The induction hypothesis states that the algorithm $\mathcal{B}^k$ only needs access to randomness at most $f_k(\epsilon,\delta)$ calls to the base simulator $\mathcal{B}$.

Now follows the proof that $\mathcal{B}^{k+1}$ requires at most $f_{k+1}(\epsilon,\delta)$ calls to the base simulator. Consider algorithm 7 for $\mathcal{B}^{k+1}$, where $k \geq 2$. First $\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})$ calls are made to the base simulator $\mathcal{B}$, then $\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}) \times T$ calls to $\mathcal{B}^k$ and at the end one call to $\mathcal{B}^k$. By the induction hypothesis, we know that $\mathcal{B}^k$ has at most $f_k(\epsilon,\delta)$ calls to the base simulator. Hence the amount of calls to the base simulator is bounded by

$$\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}) + \overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}) \times T \times f_k\Big(\frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})T}\Big) + f_k(\frac{\epsilon}{2},\frac{\delta}{2}), \tag{4.73}$$

[6] Triangle inequality: $|a+b| \leq |a| + |b|$

which is bounded by

$$\leq \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4}) \times (T+2) \times f_k\Big(\frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})T}\Big). \tag{4.74}$$

The function $f_{k+1}(\epsilon, \delta)$ bounds this due to Lemma 7, which completes this part of the induction. This proves the statement regarding the amount of calls to the base simulator
Next follows the proof of the computational cost statement. Assume that $(C+G+1)f_k(\epsilon, \delta)$ bounds the computational costs of $\mathcal{B}^k$. The proof that is also holds for $k+1$ follows.
In each of the $\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})$ iterations in the $\mathcal{B}^{k+1}$ algorithm, the following computations occur. First, one call to $\mathcal{B}(t, \eta)$ which has C computational cost. Second, T calls are made to $\mathcal{B}^k$ and by the induction hypothesis we know that each has computational cost $(C+G+1) \times f_k(\frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})T})$. Third, the minimum of a T length vector is computed. Hence the total amount of computational costs per iteration is bounded by

$$C + (C+G+1) \times T \times f_k\Big(\frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})T}\Big) + T. \tag{4.75}$$

The remaining costs of the algorithm are the following. One call is made to $\mathcal{B}^k$ with cost $(C+G+1)f_k(\frac{\epsilon}{2}, \frac{\delta}{2})$. Finally, the algorithm computes the average, and after that a subtraction is done, costing $\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4}) + 1$. The total computational cost is therefore bound by

$$\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4}) \times \Big(C + (C+G+1) \times T \times f_k\Big(\frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})T}\Big) + T\Big) + (C+G+1)f_k(\frac{\epsilon}{2}, \frac{\delta}{2}) + \overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4}) + 1. \tag{4.76}$$

This is bound by

$$\leq (C+G+1) \times (\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4}) + 1) \times (T+2) \times f_k\Big(\frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4}, \frac{\delta}{4})T}\Big). \tag{4.77}$$

Lemma 7 bounds this by $(C+G+1)f_{k+1}(\epsilon, \delta)$ which completes the induction.
This completes the proof of Lemma 9. □

Theorem 11 shows the computational costs, calls to randomness and quality of the approximation D_k of algorithm $\hat{\mathcal{B}}^k$.

Theorem 11. *Suppose w.p.1 $Z_t \in [0,1]$ for all $t \in [1,T]$. Then for all $k \geq 1$, there exists a randomized algorithm $\hat{\mathcal{B}}^k$ which takes as input any $\epsilon, \delta \in (0,1)$, and achieves the following. In total computational time at most $(C+G+1)f_{k+1}(\epsilon, \delta)$, and with only access to randomness at most $f_{k+1}(\epsilon, \delta)$ calls to the base simulator $\mathcal{B}$, returns a random number X satisfying* $\mathsf{P}(|X - D_k| \leq \epsilon) \geq 1 - \delta$.

Chapter 5

MDP as an optimal stopping problem

Recall that Chapter 3 introduced MDP, as well as several exact algorithms and an approximation method. The chapter discusses some difficulties of MDP and the reason why exact methods generally do not work. This difficulty raised the need for approximation methods, but these approximation methods generally do not have an approximate guarantee. This chapter discusses a way to obtain approximate guarantees related to MDP, using an optimal stopping problem. We do this by creating an optimal stopping problem from an MDP, denoted by OS-MDP. Hence, the bounds obtained in Chapter 4 apply on OS-MDP.
Section 5.1 introduces OS-MDP. Section 5.2 discusses policy iteration in the OS-MDP setting. Section 5.3 uses the value iteration algorithm, and finally, section 5.4 applies ADP to OS-MDP.

5.1 OS-MDP model: stochastic process over policy space

First, some preliminaries. Recall the definition of a Markov chain (Definition 2) and that an MDP with a fixed policy becomes a Markov chain. Additionally, recall that π^n is a policy in iteration n and that Π is the set of all policies. The value of a Markov chain MC is denoted by $G(MC)$, see equation (3.2). Let $MC(\pi)$ denote the Markov chain obtained from fixing policy π in the MDP.
Next, construct the stochastic process over the policy space. Let $\pi_i, \pi_j \in \Pi$ be two arbitrary policies and denote U as a transition kernel. Then $U(\pi_i, \pi_j)$ denotes the probability to go from π_i to π_j. The transition kernel U can be written as a $|\Pi| \times |\Pi|$ matrix. Consequently, U defines a Markov chain with state space Π. Let this Markov chain be defined as $\hat{\pi} = \{\pi^n, n \geq 1\}$. Denote $\hat{\pi}^N = \{\pi^n, 1 \leq n \leq N\}$. The value $G(MC(\pi))$ denotes the costs for state $\pi \in \Pi$. Define

$$OPT = \inf_{\tau 2 \mathcal{T}} \mathsf{E}\Big[G(MC(\pi^\tau))\Big], \tag{5.1}$$

where τ is a stopping time and $\mathcal{T}$ is the set of stopping times bounded on $[1, |\Pi|]$. Define $OPT(N) = \inf_{\tau 2 \mathcal{T}_N} \mathsf{E}[Z_\tau]$, where $\mathcal{T}_N$ denotes the set of all stopping times bounded on $[1, N]$. This completes the formulation of OS-MDP.

The objective OPT uses the entire interval while $OPT(N)$ uses the interval $[1, N]$. The policy obtained at τ and value OPT are not necessarily optimal for the MDP. Instead, OPT outputs the costs of the Markov chain corresponding to the policy obtained at stopping time τ. It depends on the method used whether this policy will be optimal for the MDP or not.
Observe that $\hat{\pi}$ is a Markov chain over the policy space, and $MC(\pi)$ is a Markov chain over the state space S going through time t in the original MDP. For the remainder of the chapter, define $Z_n = G(MC(\pi^n))$. Recall Definition 6 and denote $Z_n^k(N)$ as Z_n^k where n is bounded on $[1, N]$. Recall the definition of E_K, repeated here for clarity:

$$E_K = \sum_{k=1}^{K} \mathsf{E}\Big[\min_{n \in [1,N]} Z_n^k(N)\Big]. \tag{5.2}$$

Let $E_K(N)$ denote E_K where n is bounded on $[1, N]$. Next, define the regret term $\overline{Z}_n^k = Z_n^k(N) - \mathsf{E}[\min_{i \in [1,M]} Z_i^k(M)|\mathcal{F}_n]$ for $k \geq 1$.

Definition 8. *Define*

$$\overline{E}_K(N, M) = \sum_{k=1}^{K} \mathsf{E}\Big[\min_{n \in [1,N]} \overline{Z}_n^k(N, M)\Big]. \tag{5.3}$$

The following corollary states a general result for the error between $OPT(N)$ and OPT using the approximations $E_K(N)$ and $E_K(|\Pi|)$.

Theorem 12. *If $Z_n \in [0, 1]$ for all $n \in [1, |\Pi|]$ and $1 \leq N < |\Pi| < \infty$, then for any $K \geq 1$*

$$\overline{E}_K(N, |\Pi|) - \frac{1}{K+1} \leq OPT(N) - OPT \leq \overline{E}_K(N, |\Pi|) + \frac{1}{K+1} \tag{5.4}$$

Proof. Recall Theorem 7 which applied on $OPT(N)$ and OPT yields

$$OPT(N) \leq E_K(N) + \frac{1}{K+1} \quad \forall K \geq 1, \tag{5.5}$$

and

$$OPT \leq E_K(|\Pi|) + \frac{1}{K+1} \quad \forall K \geq 1. \tag{5.6}$$

Recall Theorem 6 which states that $OPT = \sum_{k=1}^{1} \mathsf{E}[\min_{n \in [1,j \Pi j} Z_n^k(|\Pi|)]$. Additionally, Lemma 3 states that Z_n^k is non-negative. Therefore by definition of E_K

$$OPT \geq E_K(|\Pi|) \quad \forall K \geq 1, \tag{5.7}$$

and

$$OPT(N) \geq E_K(N) \quad \forall K \geq 1. \tag{5.8}$$

Combining inequalities (5.5) and (5.7) yields

$$OPT(N) - OPT \leq \frac{1}{K+1} + E_K(N) - E_K(\Pi) \quad \forall K \geq 1. \tag{5.9}$$

Using inequalities (5.6) and (5.8) we obtain

$$OPT(N) - OPT \geq \frac{-1}{K+1} + E_K(N) - E_K(\Pi) \quad \forall K \geq 1. \tag{5.10}$$

Consider

$$E_K(N) - E_K(|\Pi|) = \sum_{k=1}^{K} \Big(\mathsf{E}[\min_{n \in [1,N]} Z_n^k(N)] - \mathsf{E}[\min_{i \in [1,j\Pi j]} Z_i^k(|\Pi|)] \Big). \tag{5.11}$$

Using the law of total expectation[1] and setting the random variable Y to $\mathcal{F}_t$ for some $t \in [1, |\Pi|]$, allows to rewrite equation (5.11) to

$$E_K(N) - E_K(|\Pi|) = \sum_{k=1}^{K} \Big(\mathsf{E}[\min_{n \in [1,N]} Z_n^k(N)] - \mathsf{E}[\mathsf{E}[\min_{i \in [1,j\Pi j]} Z_i^k(|\Pi|)|\mathcal{F}_t]] \Big). \tag{5.12}$$

Rewriting equation (5.12) using $\mathsf{E}[X + Y] = \mathsf{E}[X] + \mathsf{E}[Y]$ for arbitrary random variables X and Y and independence in n of the right term, yields

$$E_K(N) - E_K(|\Pi|) = \sum_{k=1}^{K} \Big(\mathsf{E}[\min_{n \in [1,N]} \Big(Z_n^k(N) - \mathsf{E}[\min_{i \in [1,j\Pi j]} Z_i^k(|\Pi|)|\mathcal{F}_t] \Big)] \Big). \tag{5.13}$$

Observe that $\{\mathsf{E}[\min_{i \in [1,j\Pi j]} Z_i^k(|\Pi|)|\mathcal{F}_t], t \geq 1\}$ is a martingale by Proposition 1. Recall equation (4.5), which states that the martingale $\{Y_t, t \geq 1\}$ has $\mathsf{E}[Y_t] = \mathsf{E}[Y_1]$. Set $t = n$, which yields

$$E_K(N) - E_K(|\Pi|) = \sum_{k=1}^{K} \Big(\mathsf{E}[\min_{n \in [1,N]} \Big(Z_n^k(N) - \mathsf{E}[\min_{i \in [1,j\Pi j]} Z_i^k(|\Pi|)|\mathcal{F}_n] \Big)] \Big) \tag{5.14}$$

$$= \overline{E}_K(N, |\Pi|) \tag{5.15}$$

Combining inequalities (5.9) and (5.10) with equality (5.15) yields inequality (5.4), which completes the proof of Theorem 12. □

The term $\overline{E}_K(N, |\Pi|)$ is the approximation of the error between $OPT(N)$ and OPT. Recall that section 4.5 introduced the algorithm to approximate E_K for a general optimal stopping problem. Observe that $\overline{E}_K(N, |\Pi|)$ consists of terms that can be estimated using algorithms $\mathcal{B}^K$ and $\hat{\mathcal{B}}^K$, see algorithms 7 and 8 respectively. Algorithm 10, denoted by $\overline{\mathcal{B}}^K$, is a sketch of the algorithm to approximate the terms of $\overline{E}_K(N, M)$, which uses algorithm 9, denoted by $\tilde{\mathcal{B}}^K$. The calls to the base simulator $\mathcal{B}$ correspond to an instance of $\hat{\pi}$.

There are many possibilities to design the transition matrix U. For example completely random, implying that every policy has equal probability after every iteration. Other options include using MDP solution methods to define U. The upcoming sections discuss the implications of Theorem 12 for policy iteration, value iteration and ADP. Each of these solution methods include a defined transition matrix U.

5.2 Policy iteration

Recall the policy iteration algorithm and relevant results from section 3.3.4. Policy iteration outputs an optimal policy π . The cardinality $|\Pi|$ bounds the amount of iterations, since the algorithm never visits a policy twice unless it is the optimal policy, in which case the algorithm terminates [24].

[1] Law of total expectation: $\mathsf{E}[X] = \mathsf{E}[\mathsf{E}[X|Y]]$ for any random variable Y on the same probability space.

Algorithm 9: Approximate $\overline{Z}_t^k(N,M)$ for given instance η.

Result: ϵ-approximation to $\overline{Z}_n^k(N,M)$ given η w.p. at least $1-\delta$
Algorithm $\tilde{\mathcal{B}}^k(n,\eta,\epsilon,\delta)$;
Create a length-$\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})$ vector A^0 ;
for *i=1 to* $\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})$ do
  Generate an independent call to $\mathcal{B}(n,\eta)$ and store in D by M matrix A^1 ;
  Create a length-M vector A^2 ;
  for *j=1 to M* do
    Generate an independent call to $\mathcal{B}^k\left(j, A^1_{[j]}, \frac{\epsilon}{4}, \frac{\delta}{4\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})M}\right)$ and store in A^2_j ;
  end
  Compute the minimum value of A^2 and store in A^0_i ;
end
Generate an independent call to $\mathcal{B}^k(n,\eta,\frac{\epsilon}{2},\frac{\delta}{2})$ and store as variable A^3 ;
Return $A^3 - (\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{4},\frac{\delta}{4})} A^0_i$;

Algorithm 10: Approximate $\mathsf{E}[\min_{n2[1,N]}\overline{Z}_n^k(N,M)]$

Result: ϵ-approximation to $\mathsf{E}[\min_{n2[1,N]}\overline{Z}_n^k(N,M)]$ w.p. at least $1-\delta$
Algorithm $\overline{\mathcal{B}}^k(\epsilon,\delta)$;
Create a length-$\overline{N}(\frac{\epsilon}{2},\frac{\delta}{2})$ vector A^0 ;
for *i=1 to* $\overline{N}(\frac{\epsilon}{2},\frac{\delta}{2})$ do
  Generate an independent call to $\mathcal{B}(0,\emptyset)$ and store in D by N matrix A^1 ;
  Create a length-N vector A^2 ;
  for *j=1 to N* do
    Generate an independent call to $\tilde{\mathcal{B}}^k\left(j, A^1_{[j]}, \frac{\epsilon}{2}, \frac{\delta}{2\overline{N}(\frac{\epsilon}{2},\frac{\delta}{2})N}\right)$ and store in A^2_j ;
  end
  Compute the minimum value of A^2 and store in A^0_i ;
end
Return $(\overline{N}(\frac{\epsilon}{2},\frac{\delta}{2}))^{-1}\sum_{i=1}^{\overline{N}(\frac{\epsilon}{2},\frac{\delta}{2})} A^0_i$;

Next, construct Markov chain $\hat{\pi}$ for policy iteration. Recall the definition of Π^n_{PI} from section 3.3.4, thus Π^n_{PI} denotes the set of best improving policies that are as similar as possible to π^n. Define U as

$$U(\pi^n,\pi_j) = \begin{cases} \frac{1}{j\Pi^n_{PI}j} & \text{if } \pi_j \in \Pi^n_{PI}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.16}$$

Assume that $|\Pi^n_{PI}| = 1$ for all $n \in [1,|\Pi|]$, thus always a single best improving policy, unless stated otherwise.
Consider the optimal stopping problem defined by Markov chain $\hat{\pi}$. Observe that when considering $\hat{\pi}^N$ for $1 \le N < |\Pi|$, the policy found at N is not necessarily optimal. But the best

policy for the Markov chain is always obtained at N, by Theorem 2, hence $\tau = N$ is optimal. Next, consider Theorem 12 in the setting of policy iteration. Here OPT is the value corresponding to an optimal policy for the MDP and $OPT(N)$ uses the policy found at N, since the stopping problem defined on $[1, N]$ has as optimal stopping time $\tau = N$. Therefore Theorem 12 gives an estimate of the difference between π^N and π , which is an estimate of the error of terminating policy iteration early.

Corollary 2. *Let $|\Pi_{PI}^n| = 1$ for all $n \geq 1$ and $1 \leq N < |\Pi|$. Let $\hat{\pi}$ the Markov chain created by policy iteration and $\hat{\pi}^N$ the N steps Markov chain using policy iteration. Then*

$$OPT(N) - OPT = \overline{E}_1(N, |\Pi|) \tag{5.17}$$

Proof. Let $a \in [1, |\Pi|]$. First, show that $OPT(a) = \mathsf{E}[Z_a]$ for any $a \in [1, |\Pi|]$. Recall that $OPT(a) = \inf_{\tau 2 T} \mathsf{E}[Z_\tau]$. The construction of U yields that the best policy is always obtained at a, therefore $\tau = a$ is optimal. This implies $OPT(a) = \mathsf{E}[Z_a]$, which completes the first part of the proof.
Second is to show that $E_K(a) = \mathsf{E}[Z_a]$. Recall the definition of Z_n^k at Definition 6. Consider $E_1(a) = \mathsf{E}[\min_{n 2 [1,a]} Z_n]$. The construction of U yields that $n = a$ obtains the best policy, and therefore $E_1(a) = \mathsf{E}[Z_a]$. Next is to show that $\mathsf{E}[\min_{n 2 [1,a]} Z_n^k] = 0$ for $k \geq 2$. Induction shows this. First, consider $k = 2$ and use Definition 6 to obtain

$$\mathsf{E}[\min_{n 2 [1,a]} Z_n^2] = \mathsf{E}[\min_{n 2 [1,a]} (Z_n - \mathsf{E}[\min_{i 2 [1,a]} Z_i | \mathcal{F}_n])]. \tag{5.18}$$

Since $|\Pi^n| = 1$ for all $n \geq 1$, there is an unique policy at every iteration n. Therefore the term $\mathsf{E}[\min_{i 2 [1,a]} Z_i | \mathcal{F}_n] = \mathsf{E}[Z_a]$, since the filtration has no influence. This implies

$$\mathsf{E}[\min_{n 2 [1,a]} Z_n^2] = \mathsf{E}[\min_{n 2 [1,a]} [Z_n - \mathsf{E}[Z_a]]] = 0. \tag{5.19}$$

This proves the first part of the induction.
Assume that $\mathsf{E}[\min_{n 2 [1,a]} Z_n^k] = 0$ holds for some $K \geq 2$, then show that $\mathsf{E}[\min_{n 2 [1,a]} Z_n^{K+1}] = 0$. Applying Definition 6 gives

$$\mathsf{E}[\min_{n 2 [1,a]} Z_n^{K+1}] = \mathsf{E}[\min_{n 2 [1,a]} Z_n^K - \mathsf{E}[\min_{i 2 [1,a]} Z_i^K | \mathcal{F}_n]]. \tag{5.20}$$

Observe that by $|\Pi_{PI}^n| = 1$ we obtain $\mathsf{E}[\min_{i 2 [1,a]} Z_i^K | \mathcal{F}_n] = \mathsf{E}[\min_{i 2 [1,a]} Z_i^K]$ for all $n \in [1, a]$. The induction hypothesis states that $\mathsf{E}[\min_{i 2 [1,a]} Z_i^K] = 0$ and therefore

$$\mathsf{E}[\min_{n 2 [1,a]} Z_n^{K+1}] = \mathsf{E}[\min_{n 2 [1,a]} Z_n^K - 0] = 0. \tag{5.21}$$

This completes the induction. Using the previous results and the definition of $\overline{E}_1(N, M)$ yields

$$OPT(N) - OPT = E_1(N) - E_1(\Pi) = \overline{E}_1(N, |\Pi|). \tag{5.22}$$

This completes the proof of Corollary 2. □

Recall $MC(\pi)$ denotes the Markov chain obtained by fixing policy π in the MDP and that $G(MC)$ computes the value of Markov Chain MC. The next corollary makes Corollary 2 more specific.

Corollary 3. *Let $|\Pi_{PI}^n| = 1, \forall n \geq 1$ and let $\hat{\pi}$ denote the Markov chain obtained by policy iteration. Let $G(MC(\pi)) \in [0,1]$ for all $\pi \in \Pi$. Then*

$$OPT(N) - OPT = \mathsf{E}[G(MC(\pi^N))] - \mathsf{E}[G(MC(\pi^*))]. \tag{5.23}$$

Proof. Corollary 2 states that $OPT(N) - OPT = \overline{E}_1(N, |\Pi|)$. By the definition of $\overline{Z}_t^k$ and Definition 6,

$$\overline{E}_1(N, \Pi) = \mathsf{E}[\min_{n \in [1,N]} \Big(Z_n(N) - \mathsf{E}[\min_{i \in [1,j \Pi j]} Z_i(|\Pi|) | \mathcal{F}_n] \Big)]. \tag{5.24}$$

Since $|\Pi_{PI}^n| = 1$, there is always a single best improving policy and hence the filtration $\mathcal{F}_n$ has no influence on the right term. Additionally, the term $Z_n = G(MC(\pi^n))$ denotes the value of the Markov chain obtained by applying π^n to the MDP. Recall from Theorem 2 that $V^{n+1} \leq V^n$ and as a consequence $G(MC(\pi^{n+1})) \leq G(MC(\pi^n))$. Therefore the minimum $Z_i, i \in [1, |\Pi|]$ will be obtained at $|\Pi|$, thus obtaining the optimal policy. Therefore

$$OPT(N) - OPT = \mathsf{E}[G(MC(\pi^N))] - \mathsf{E}[G(MC(\pi^*))]. \tag{5.25}$$

This completes the proof of Corollary 3. □

Corollary 3 states that the error between $OPT(N)$ and OPT is the expected value obtained at N minus the expected value of the optimal policy. This is not a surprise, since policy iteration improves in every iteration and there is no randomness involved.

5.3 Value iteration

The value iteration algorithm is introduced in section 3.3.3. Recall that the value iteration algorithm computes an $\gamma-$optimal policy and that updating the values of each state, until the stopping criteria is satisfied, achieves this. The sequence $\{V^n, n \geq 1\}$ converges in norm to V^*, by Theorem 1, where V^* is the value of each state corresponding to the optimal policy π^*. Any value vector V^n has a corresponding policy, denoted by π^n. Observe that $G(MC(\pi^n))$ has no specific relation to $G(MC(\pi^{n+1}))$ as was the case in policy iteration. This implies that the policy π^n can be better then π^{n+1} [24]. The parameter M is used instead of $|\Pi|$, where $N < M < \infty$.

The amount of iterations is finite, by Theorem 1, but is unknown at the start of the algorithm. Recall the definition of Π_{VI}^n in section 3.3.3, which denotes the set of policies corresponding to value vector V^n. Assume $|\Pi_{VI}^n| = 1$. Define the transition matrix U as

$$U(\pi^n, \pi_j) = \begin{cases} 1 & \text{if } \pi_j \in \Pi_{VI}^{n+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.26}$$

Corollary 4 applies Theorem 1 on Theorem 12.

Corollary 4. *Let $\hat{\pi}$ denote the Markov chain obtained by value iteration. Let $G(MC(\pi)) \in [0,1]$ for all $\pi \in \Pi$ and $|\Pi_{VI}^n| = 1$. Then*

$$OPT(N) - OPT(M) \leq \frac{1}{2} + 2T\frac{\lambda^N + \lambda^M}{1-\lambda}||V_1 - V_0|| + \min_{i \in [1,M]}(T||V_{\pi^i}|| - Z_i). \tag{5.27}$$

Proof. Recall equation (3.11). Additionally recall the definition of the vector norm $||V|| = \sup_{s \in S}|V(s)|$.
Next consider $\overline{E}_K(N,M) = \sum_{k=1}^{K} \mathsf{E}[\min_{n \in [1,N]} \overline{Z}_n^k(N,M)]$ and recall the definition of $\overline{Z}_t^k$. Let $k=1$, which yields

$$\overline{E}_1(N,M) = \mathsf{E}[\min_{n \in [1,N]} \overline{Z}_n^1(N,M)] \tag{5.28}$$

$$= \mathsf{E}\Big[\min_{n \in [1,N]} \Big(Z_n(N) - \mathsf{E}[\min_{i \in [1,M]} Z_i|\mathcal{F}_n]\Big)\Big]. \tag{5.29}$$

Recall that the MDP has T time steps. This implies, given some policy π^n, that $Z_n \leq T||V_{\pi^n}||$. This holds because visiting the state with the highest costs T times is worse or equal then any other sequence of states visited. Hence a bound on equation (5.29) is

$$\leq \mathsf{E}\Big[\min_{n \in [1,N]} \Big(T||V_{\pi^n}|| - \mathsf{E}[\min_{i \in [1,M]} Z_i|\mathcal{F}_n]\Big)\Big]. \tag{5.30}$$

The triangle inequality states $||x+y|| \leq ||x|| + ||y||$. Therefore $||V_{\pi^n}|| = ||V_{\pi^n} - V_\pi \ + V_\pi \ || \leq ||V_{\pi^n} - V_\pi \ || + ||V_\pi \ ||$. Applying this to inequality (5.30) yields

$$\leq \mathsf{E}\Big[\min_{n \in [1,N]} \Big(T||V_{\pi^n} - V_\pi \ || - \mathsf{E}[\min_{i \in [1,M]} (Z_i - T||V_\pi \ ||)|\mathcal{F}_n]\Big)\Big]. \tag{5.31}$$

Next, apply Theorem 1 to the above equation to obtain

$$\leq \mathsf{E}\Big[\min_{n \in [1,N]} \Big(T\frac{2\lambda^n}{1-\lambda}||V^1 - V^0|| - \mathsf{E}[\min_{i \in [1,M]} (Z_i - T||V_\pi \ ||)|\mathcal{F}_n]\Big)\Big]. \tag{5.32}$$

Note that the filtration $\mathcal{F}_n$ and the expectation are removable, since $|\Pi_{VI}^n| = 1$ for all n. Additionally, observe that for $\lambda \in (0,1)$ the left term is minimised at N. Therefore, equation rewrites (5.32) to

$$= T\frac{2\lambda^N}{1-\lambda}||V^1 - V^0|| - \min_{i \in [1,M]}(Z_i - T||V_\pi \ ||). \tag{5.33}$$

Let $||V_\pi \ || = ||V_\pi \ + V_{\pi^i} - V_{\pi^i}||$, then rearranging terms in (5.33) and applying the triangle inequality yields

$$\leq T\frac{2\lambda^N}{1-\lambda}||V^1 - V^0|| + T(||V_\pi \ || - ||V_{\pi^M}||) - \min_{i \in [1,M]}(Z_i - T||V_{\pi^i}||). \tag{5.34}$$

Theorem 1 bounds equation (5.34) by

$$\leq 2T\frac{\lambda^N + \lambda^M}{1-\lambda}||V^1 - V^0|| + \min_{i \in [1,M]}(T||V_{\pi^i}|| - Z_i). \tag{5.35}$$

This completes the proof of Corollary 4. □

5.4 Approximate dynamic programming

Recall ADP from section 3.4.1. The idea of ADP is to approximate the value of each state, denoted by $\overline{V}^n$ for $n \geq 1$. Similar to value iteration, discussed in the previous section, a value vector has a corresponding policy. Define the set Π^n_{ADP} as $\arg\min_{\pi 2 \Pi}\{C_\pi + \lambda P_\pi \overline{V}^n\}$, which is the set of optimal policies given value vector $\overline{V}^n$. The method in which $\overline{V}^n$ is computed can vary. For example, the standard ADP algorithm 5, or the basis function approach, as discussed in section 3.4.2, are both possibilities. Define U as

$$U(\pi^n, \pi_j) = \begin{cases} \frac{1}{j\Pi^{n+1}_{ADP}j} & \text{if } \pi_j \in \Pi^{n+1}_{ADP}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.36}$$

In policy iteration and value iteration, it is certain that the set Π^n would at some point contain optimal policies or γ-optimal policies respectively. This is not the case for ADP, hence OPT is not the optimum for the MDP.

Let M bound the amount of iterations used by ADP, because ADP can visit policies multiple times. Denote $OPT(M)$ instead of OPT to di□er between the optimum for ADP and MDP respectively. The ADP algorithm finds $OPT(M)$ when it terminates after M iterations.

Theorem 12 gives an estimate of the error using π^n versus π^m, where $n \in [1, N]$ and $m \in [1, M]$ for $N < M$. Algorithm 10 approximates the di□erence between the values obtained by π^n and π^m.

Chapter 6

Results for The Game

This chapter applies the results obtained in Chapter 5 to The Game. This is done for illustration. Section 3.5.2 introduces The Game and models it as an MDP. Additionally, 3.5.3 gives a description of the application of ADP to The Game.
Section 6.1 applies OS-MDP to The Game for the three solution methods; policy iteration, value iteration, and ADP. Section 6.2 contains the results of the ADP method applied on The Game. Finally, section 6.3 gives a numerical example to bound The Game using OS-MDP.

6.1 Applying OS-MDP to The Game

This section uses the bounds obtained in Chapter 5 and relates them to The Game. Recall equation (3.41) and define the objective of the OS-MDP for The Game as

$$OPT_{TG}(N) = \inf_{\tau 2 T} \Big(98 - \mathsf{E}\Big[\sum_{t=1}^{T}\sum_{c=2}^{99}\sum_{p=1}^{4} x_t^{\pi^\tau}(p,c)\Big]\Big), \tag{6.1}$$

where $x_t^\pi(p,c)$ denotes whether policy π plays card c on pile p in turn t. Applying the results of Chapter 5 require $OPT_{TG}(N) \in [0,1]$, obtained by dividing the objective value by 98. Additionally, define Z_n as the expected value obtained by policy π^n. Therefore

$$Z_n = 1 - \frac{1}{98}\mathsf{E}\Big[\sum_{t=1}^{T}\sum_{c=2}^{99}\sum_{p=1}^{4} x_t^{\pi^n}(p,c)\Big]. \tag{6.2}$$

6.1.1 Policy iteration

To determine the error of stopping early in policy iteration for The Game, compute the error $OPT_{TG}(\pi^N) - OPT_{TG}$. Applying Corollary 3 and rearranging the terms yields

$$0 \leq OPT_{TG}(N) - OPT_{TG}(|\Pi|) = \frac{1}{98}\Big(\mathsf{E}\Big[\sum_{t=1}^{T}\sum_{c=2}^{99}\sum_{p=1}^{4} x_t^{\pi}\ (p,c)\Big] - \mathsf{E}\Big[\sum_{t=1}^{T}\sum_{c=2}^{99}\sum_{p=1}^{4} x_t^{\pi^n}(p,c)\Big]\Big). \tag{6.3}$$

Unfortunately, equation (6.3) is not particularly useful, since it simply states the difference between the two expected values of the policies.

6.1.2 Value iteration

Recall Corollary 4, which bounds the error using value iteration. The parameter $T = 49$ in The Game. Additionally, fill in Z_i using equation (6.2). Assume that $V_0 = 0$. Therefore, $||V_1 - V_0|| \leq \frac{7}{98}$, which denotes the largest possible jump in value from one instant reward. Combining this yields

$$OPT_{TG}(N) - OPT_{TG}(M) \leq \frac{1}{2} + \frac{7}{2}\frac{\lambda^N + \lambda^M}{1-\lambda} + \min_{i \in [1,M]} \Big(49||V_{\pi^i}|| - 1 + \frac{1}{98}\mathsf{E}\Big[\sum_{t=1}^{T}\sum_{c=2}^{99}\sum_{p=1}^{4} x_t^{\pi^i}(p,c)\Big]\Big). \tag{6.4}$$

Note that $OPT_{TG}(N) - OPT_{TG}(M) \in [0,1]$. Therefore, inequality (6.4) needs $\frac{7}{2}\frac{\lambda^N + \lambda^M}{1\ \lambda} < \frac{1}{2}$. For example, for $\lambda = 0.9$ and $M = 50$ requires $N = 45$. And $\lambda = 0.99$ and $M = 1000$, requires at least $N = 656$. This observation excludes the impact of the minimisation term in equation (6.4), thus larger M and N are probably required.

6.1.3 ADP

Recall Definition 8. Apply equation (6.2) to $\overline{E}_K(N,M)$, and rearrange the terms to obtain

$$\overline{E}_K(N,M) = \frac{1}{98}\mathsf{E}\Big[\min_{n \in [1,N]} \Big(\mathsf{E}[\min_{i \in [1,M]} \sum_{t=1}^{T}\sum_{c=2}^{99}\sum_{p=1}^{4} x_t^{\pi^i}(p,c)]|\mathcal{F}_n] - \mathsf{E}[\sum_{t=1}^{T}\sum_{c=2}^{99}\sum_{p=1}^{4} x_t^{\pi^n}(p,c)]\Big)\Big] + \sum_{k=2}^{K}\mathsf{E}\Big[\min_{n \in [1,N]} \overline{Z}_n^k(N,M)\Big]. \tag{6.5}$$

As illustration, let $k = 1$. Combining Theorem 12 and equation (6.5), yields the following upper and lower bound,

$$OPT_{TG}(N) - OPT_{TG}(M) \leq (\geq)\frac{1}{98}\mathsf{E}\Big[\min_{n \in [1,N]} \Big(\mathsf{E}[\min_{i \in [1,M]} \sum_{t=1}^{T}\sum_{c=2}^{99}\sum_{p=1}^{4} x_t^{\pi^i}(p,c)]|\mathcal{F}_n] - \mathsf{E}[\sum_{t=1}^{T}\sum_{c=2}^{99}\sum_{p=1}^{4} x_t^{\pi^n}(p,c)]\Big)\Big] + (-)\frac{1}{2}. \tag{6.6}$$

6.2 Approximating The Game using ADP

This section applies the approximation method described in section 3.5.3 for The Game. This sections also tests several di□erent initial settings and sets of features. The updating parameter κ for the recursive least squares method (see appendix) is set on either 0, implying stationary, or 0.5, implying non-stationary. The exploration and exploitation settings N_1 and N_2 are set such that $N_1 \times N_2$ equals 300. All runs use the same set of sample paths. Table 6.1 contains all settings for each run. Additionally, the initial feature weights θ_f^0 are always initialised on 1 for every feature $f \in F$. The settings of each run are as follows;

R1: Initial run with a set of features which seemed reasonable. Non-stationary data.

R2: A larger set of features with stationary data.

R3: The features 'Jumping back pairs' and both 'abs dif' features are removed. The absolute difference between piles feature is obsolete, which we will explain in section 6.2.1. Non-stationary data.

R4: Same as R3, except stationary data.

R5: Similar to R3, however without the 'Sum min dif hand' feature, because it had a small feature weight and hence little impact.

R6: Same as R5, except stationary data.

R7: Same as R3, but instead a more exploitation heavy approach is taken. Therefore N_2 is increased and N_1 decreased.

R8: Same as R7, except stationary data.

Table 6.1: Initial settings for ADP applied on The Game. Number in brackets refer to the features in section 3.5.3.

	Name	R1	R2	R3	R4	R5	R6	R7	R8
Features	Pile 1 (1)	*	*	*	*	*	*	*	*
	Pile 2 (1)	*	*	*	*	*	*	*	*
	Pile 3 (1)	*	*	*	*	*	*	*	*
	Pile 4 (1)	*	*	*	*	*	*	*	*
	Playable space (2)	*	*	*	*	*	*	*	*
	Amount of cards in play (3)	*	*	*	*	*	*	*	*
	Abs dif 1 and 2 (5)	*	*						
	Abs dif 3 and 4 (5)	*	*						
	Jumping back pairs (7)		*						
	Sum min dif (8)		*	*	*			*	*
Initial settings	N_1	50	50	50	50	50	50	15	15
	N_2	6	6	6	6	6	6	20	20
	κ (0 means stationary)	0.5	0	0.5	0	0.5	0	0.5	0

6.2.1 Analysis of different settings

Every run yields feature weights of the final iteration, given in table 6.2. The feature weights determine the policy used for The Game. One iteration of ADP takes approximately 5 minutes. Hence 300 iterations take close to 25 hours.
Recall the definition of OPT_{TG} from equation (3.41). The formulation of OPT_{TG} maximises the value of each state. Additionally, recall that every feature $f \in F$ satisfies $\phi_f(s) \geq 0$. Therefore, a positive feature weight implies good characteristic of the state, while a negative feature weight corresponds to a bad characteristic of the state. For example, consider R1, where $\theta_1 = -67.7$ and $\theta_2 = 10.2$. Therefore in R1, actions are preferred that reduce the

playable space on pile 1 and maximise the playable space on pile 2. Hence, playing a 90 on pile 1 is an improvement of the value of the state, while playing a 90 on pile 2 is not.
The features 'abs dif 1 and 2' and 'abs dif 3 and 4' were removed after R2. This removal is because they were obsolete. Consider R1, where the feature weight for $\theta_7 = 197$. However, observe that the separate feature values also di□er, $\theta_1 = -67.7$ and $\theta_1 = 10.2$. Therefore, assume that the number on pile 1 will be larger than on pile 2. Hence feature 'abs di□ 1 and 2' can be written as $Q_t(1) - Q_t(2)$, which implies

$$\overline{V}(s) = \theta_1 Q_t(1) + \theta_2 Q_t(2) + \theta_7(Q_t(1) - Q_t(2)) + \sum_{f2Fnf1,2,7g} \theta_f \phi_f(s) \tag{6.7}$$

$$= (\theta_1 + \theta_7)Q_t(1) + (\theta_2 - \theta_7)Q_t(2) + \sum_{f2Fnf1,2,7g} \theta_f \phi_f(s). \tag{6.8}$$

Therefore 'abs di□ 1 and 2' is obsolete when there is a strong preference for one of the two piles since features 'Pile 1' and 'Pile 2' capture this.
Several observations can be made from table 6.2 for each run. In every run, 'cards in play' has a positive influence on the value of the state, except for R1. The instance R6 is interesting since it contains two feature weights very close to zero. There are several instances with almost the same feature weights for piles 3 and 4, namely R1, R4, R5 and R6. Piles 1 and 2 do not have such similar feature weights.

Table 6.2: Feature weights for every feature after 300 iterations. Numbers between brackets correspond to the feature defined in section 3.5.3.

	R1	R2	R3	R4	R5	R6	R7	R8
Pile 1 (1)	-67.7	1.16	-2.85	5.67	31.1	-0.0664	-3.87	-4.77
Pile 2 (1)	10.2	-1.06	30.5	4.91	30.4	0.792	21.8	-2.80
Pile 3 (1)	129	0.761	-7.31	-1.18	-1.28	1.00	-73.1	-7.13
Pile 4 (1)	129	-0.633	8.37	-1.18	-1.31	1.00	-43.5	4.71
Playable space (2)	-80.1	1.08	-12.6	-1.85	16.1	-0.0962	-1.99	3.04
Cards in play (3)	-162	17.3	101	19.0	173	14.0	108	18.89
Abs dif 1 and 2 (5)	-197	0.844						
Abs dif 3 and 4 (5)	0.105	2.52						
Jumping back pairs (7)		-0.896						
Sum min dif (8)		0.383	-0.246	1.51			-0.564	3.17

6.2.2 Comparing strategies for The Game

Feature weights yield policies for The Game. This section compares these obtained policies. First, two random instances are created, which are played out by the policies and a card game expert. Then observe how the policies perform compared to the expert. Second, construct several instances where a perfect score is obtainable. These instances are then played out by the policies and compared. The instances are found in the appendix. Table 6.3 states the results.

The expert performs best in the two random instances. Policy R6 performed very well and achieved an almost equal score. All other policies are significantly worse.
The three winnable instances also show policy R6 as best. The expert did not play the constructed instances, since the expert knew the sequence. Policy R2 also performed well in these three instances. In both the random instances and winnable instances, policy R8 performed terribly.
Observe that policies R7 and R8 seem to perform worse. These two settings used a more exploitation heavy approach and less exploration. Therefore more exploration is preferred for The Game. Let us analyse policies R2, R6 and R8 more thoroughly. First, consider policy R6, which performed the best of the policies created by ADP. Features 'Pile 1' and 'Playable space' are close to zero, which implies that they have little impact on the value of a state. All other feature weights are positive. None of the other policies has this; there is always a feature with a significant negative weight. The non-negative feature weights are what sets R6 apart from the other policies. Policy R2 performed second best, even obtaining a perfect score in one winnable instance. What policies R2 and R6 have in common, is that the feature weight differences between piles 1-2 and 3-4 are rather small. For example, consider R2. Features 'Pile 1' and 'Pile 3' have feature weights $\theta_1 = 1.16$ and $\theta_3 = 0.761$ respectively. Though there is a difference, the difference in other runs is larger, for example, R1, with feature weights $\theta_1 = -67.7$ and $\theta_3 = 129$.
Policy R8 performed worse in all instances. Several negative feature weights for the individual piles imply that the policy wants to make big jumps. The relatively large feature weight for 'sum min dif' implies that a small jump from hand cards to pile cards is preferred. Therefore, the policy will refrain from playing cards with small differences and rather play cards with large differences on piles. This behaviour explains why R8 performs terribly.

Table 6.3: Amount of cards remaining after playing The Game.

	Self	R1	R2	R3	R4	R5	R6	R7	R8
Random 1	28	65	50	58	58	60	29	66	74
Random 2	38	63	60	60	57	58	41	78	80
Winnable 1	-	20	0	19	18	19	0	19	58
Winnable 2	-	68	31	69	60	63	3	77	87
Winnable 3	-	69	6	56	71	73	1	83	89

6.3 Error bound for The Game using ADP

This section discusses the approximation $\overline{E}_k(N, M)$ and whether it can be approximated efficiently for ADP. For the ADP, use the initial settings of R6. Algorithm 10 uses $N = 5$, $M = 10$ and $k = 1$. Additionally, the computation of $G(MC(\pi))$ was approximated for all policies, using 2 samples to approximate the value. Also, the parameters ϵ and δ were not assigned and instead assign $\overline{N}(\epsilon, \delta) = 2$. One test instance, using the settings of R6, is performed.

The result is $\overline{E}_k(N, M) = 0.25$, which yields the following bound

$$0 \leq OPT(N) - OPT(M) \leq 0.25. \tag{6.9}$$

The lower bound found -0.25, however it is not useful since $OPT(N) - OPT(M) \geq 0$. The difference between $OPT(N)$ and $OPT(M)$ is at maximum 0.25, which corresponds with 24 cards. Hence, running the ADP algorithm for 10 iterations instead of 5 would yield a maximum expected improvement of 24 cards. This bound has no guarantee since the error parameter ϵ, and δ were not initialised formally. This simple instance required four days of computation.

Chapter 7

Conclusions and Recommendations

7.1 Conclusions

MDPs are challenging to optimally solve in practical instances. This challenge is due to the three curses of dimensionality: size state-space, size action set and random information possibilities. The Bellman equations show this since large state-space implies many equations that require solving. Simultaneously, a large action set requires checking of each action and large random information set implies the computation of large sums.

Approximation methods are frequently used in practical instances to solve MDPs. One approximation method is ADP, which uses the Bellman equations to approximate the value of each state. Determining the quality of different ADP settings is achieved by comparing it to other approximation results or smaller MDPs instances. However, there are no approximate guarantees for ADP related to MDP.

In this thesis, we attempted to apply the results from Chen & Goldberg [10] to general MDPs. We were able to create an optimal stopping problem from an MDP by creating a stochastic process over the policy space, denoted by OS-MDP. This model creates an optimal stopping problem to which the results of [10] can be applied, mainly the approximate guarantee given in Theorem 7.

The approximate guarantees obtained were applied to the OS-MDP model created in Chapter 5. We obtained an error bound between running methods N steps versus M steps, where $N < M$. For this it was required that the MDP is normalised and formulated with a minimisation objective function. The bound is denoted by $\overline{E}_K(N, M)$ and can be approximated with error parameters ϵ, δ and with absolute difference of $\frac{1}{k+1}$ from $OPT(N) - OPT(M)$.

The OS-MDP was constructed by using policy iteration, value iteration and ADP. However, the bounds could not be applied effectively on policy iteration, as the regret terms Z_n^k for $k \geq 2$ would always be equal to 0. Therefore, the approximation that should depend on k, did not depend on k at all.

Value iteration applied to OS-MDP did yield a bound, given in Corollary 4. The bound was only from above and dependent on the discount factor λ and the MDP horizon length T.

Applying ADP to construct the OS-MDP, yields a way to determine the error of terminating ADP algorithm early, therefore determining the benefit of continuing ADP. This however,

is not an approximate guarantee related to a MDP, but an approximate guarantee only for ADP.

The Game was used as the running example throughout this thesis. Several policies were acquired by applying ADP on The Game. The basis function approach was applied to approximate the value of the post-decision state. The recursive least squares method was used with both stationary and non-stationary settings to update the feature weights.

Every setting used 300 iterations, with a combination of exploitation and exploration. The run-time was approximately 25 hours. Every obtained policy was tested for several random examples and several winnable examples. Comparing the results of the random examples shows that policy R6 plays equally good as an expert, while the other policies performed worse than R6. The winnable instances showed that again policy R6 performed best, however, policy R2 also obtained good results. Both R2 en R6 use stationary data updating, therefore, when ADP is applied to The Game, stationary data updating is preferred. Polices R1 up to R6 used $N_1 = 50$ and $N_2 = 6$, while R7 and R8 used $N_1 = 15$ and $N_2 = 20$. This change in settings resulted in R7 and R8 performing terribly and as such a more exploration heavy approach is preferable.

Finally, one OS-MDP instance was tested on The Game. The parameters ϵ and δ were not initialised, instead the number of iterations were fixed. Additionally, $N = 5$, $M = 10$ and the settings of R6 were used for the ADP script. Doing so, an upper bound of 0.25 between $OPT_{TG}(N)$ and $OPT_{TG}(M)$ was obtained. This result corresponds with an maximal improvement of 24 more cards played. Four days were required to carry out the test of this small scenario.

The goal of this thesis was to determine whether optimal stopping problems can be used to obtain an approximate guarantee for ADP related to MDP. This goal was not achieved, since no good relation between the ADP solutions and MDP solution was found.

7.2 Recommendations and future research

Several assumptions were made on the initial MDP. Namely, a finite horizon, finite-sized state-space and a finite action set in each state. Future research is needed to formulate OS-MDP for MDPs where these assumptions do not hold.

Only the approximate guarantee of Theorem 7 was applied to OS-MDP, whereas, Theorems 9 and 10 also give an approximate guarantee.

In both value iteration and policy iteration, it was assumed that there is always a single best improvement in the OS-MDP model. For future research it can be interesting to remove this assumption and see how Corollaries 2 and 4 change.

The bounds obtained for value iteration in section 5.3 can most likely be improved, as currently it only contains a bound for $k = 1$ and no lower bound. Doing so, a lower bound can be obtained, as well as a statement for larger k.

Another possible topic for future research is to apply OS-MDP on simulated-annealing since both search in the policy space. Other solution methods or heuristic approaches, for example, tabu-search or generic algorithms, can also be used as input for OS-MDP.

Algorithms 9 and 10 were not shown to have guarantees and are only given as a sketch of the application of OS-MDP. Proving the correctness of these algorithms or modifying the algorithms so that they hold, can be a possible direction for future research.
The numerical test of OS-MDP on The Game was carried out for a small instance and a few iterations. Only a small numerical test was conducted, due to the computational difficulties as even this small scenario took four days to compute. With this in mind, a smaller or different problem than The Game is required to test the boundaries numerically.

Bibliography

[1] E. H. L. Aarts and P. J. M. Laarhoven. Simulated annealing: An introduction. *Statistica Neerlandica*, 43(1):31–52, mar 1989.

[2] V. Bentkus et al. On Hoe□ding's inequalities. *The Annals of Probability*, 32(2):1650–1673, 2004. doi: 10.1214/009117904000000360.

[3] R. J. Boucherie and N. M. van Dijk, editors. *Markov Decision Processes in Practice*. Springer International Publishing, 2017.

[4] H. Burgiel. How to lose at Tetris. *The Mathematical Gazette*, 81(491):194–200, 1997. doi:10.2307/3619195.

[5] J. F. Carriere. Valuation of the early-exercise price for options using simulations and nonparametric regression. *Insurance: Mathematics and Economics*, 19(1):19–30, dec 1996.

[6] C. Fahey. Tetris. `https://www.colinfahey.com/tetris/tetris.html`, 2019 (accessed Nov 25, 2019).

[7] A. Fern, S. Yoon, and R. Givan. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research*, 25:75–118, jan 2006.

[8] S. Fianu and L. B. Davis. A Markov decision process model for equitable distribution of supplies under uncertainty. *European Journal of Operational Research*, 264(3):1101–1115, feb 2018.

[9] V. Gabillon, M. Ghavamzadeh, and B. Scherrer. Approximate dynamic programming finally performs well in the game of Tetris. In *Advances in neural information processing systems*, pages 1754–1762, 2013.

[10] D. A. Goldberg and Y. Chen. Beating the curse of dimensionality in options pricing and optimal stopping. *arXiv preprint arXiv:1807.02227*, 2018. arXiv:1807.02227.

[11] T. P. Hill and R. P. Kertz. Stop rule inequalities for uniformly bounded sequences of random variables. *Transactions of the American Mathematical Society*, 278(1):197–207, 1983. doi:10.1090/S0002-9947-1983-0697070-7.

[12] M. Horiguchi. Markov decision processes with a stopping time constraint. *Mathematical Methods of Operations Research*, 53(2):279–295, jun 2001.

[13] P. J. H. Hulshof, M. R. K. Mes, R. J. Boucherie, and E. W. Hans. Patient admission planning using approximate dynamic programming. *Flexible services and manufacturing journal*, 28(1-2):30–61, 2016. doi:10.1007/s10696-015-9219-1.

[14] S. Karlin. *A First Course in Stochastic Processes*. Academic press, 2014.

[15] C. Keerthisinghe, A. C. Chapman, and G. Verbic. PV and demand models for a Markov decision process formulation of the home energy management problem. *IEEE Transactions on Industrial Electronics*, 66(2):1424–1433, feb 2019.

[16] F. A. Longsta and E. S. Schwartz. Valuing American options by simulation: A simple least-squares approach. *Review of Financial Studies*, 14(1):113–147, jan 2001.

[17] M. S. Maxwell, M. Restrepo, S. G. Henderson, and H. Topaloglu. Approximate dynamic programming for ambulance redeployment. *INFORMS Journal on Computing*, 22(2):266–281, may 2010.

[18] M. R. K. Mes and A. P. Rivera. Approximate dynamic programming by practical examples. In R. J. Boucherie and N. M. van Dijk, editors, *Markov Decision Processes in Practice*, chapter 3, pages 63–101. Springer, 2017. doi:10.1007/978-3-319-47766-4_3.

[19] C. Novoa and R. Storer. An approximate dynamic programming approach for the vehicle routing problem with stochastic demands. *European Journal of Operational Research*, 196(2):509–515, jul 2009.

[20] M. Otten, J. Timmer, and A. Witteveen. Stratified breast cancer follow-up using a continuous state partially observable Markov decision process. *European Journal of Operational Research*, 281(2):464–474, mar 2020.

[21] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, volume 703. John Wiley & Sons, 2007. doi: 10.1002/9781118029176.

[22] W. B. Powell. Approximate dynamic programming: Lessons from the field. In *2008 Winter Simulation Conference*, pages 205–214. IEEE, 2008. doi:10.1109/WSC.2008.4736069.

[23] W. B. Powell. What you should know about approximate dynamic programming. *Naval Research Logistics (NRL)*, 56(3):239–249, 2009. doi:10.1002/nav.20347.

[24] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014. doi:10.1002/9780470316887.

[25] L. C. G. Rogers. Monte Carlo valuation of American options. *Mathematical Finance*, 12(3):271–286, 2002. doi:10.1111/1467-9965.02010.

[26] S. M. Ross. *Stochastic Processes*. John Wiley & Sons, 1996.

[27] S. M. Ross. *Introduction to Probability Models*. Academic press, 2014. doi:10.1016/C2013-0-11417-1.

[28] E. Saha and P. K. Ray. Modelling and analysis of healthcare inventory management systems. *OPSEARCH*, 56(4):1179–1198, sep 2019.

[29] S. Sanner and C. Boutilier. Approximate linear programming for first-order MDPs. *arXiv preprint arXiv:1207.1415*, 2012.

[30] B. Scherrer, V. Gabillon, M. Ghavamzadeh, and M. Geist. Approximate modified policy iteration. *arXiv preprint arXiv:1205.3054*, 2012. arXiv:1205.3054.

[31] J. Schoenmakers, J. Zhang, and J. Huang. Optimal dual martingales, their analysis, and application to new algorithms for Bermudan products. *SIAM Journal on Financial Mathematics*, 4(1):86–116, 2013. doi:10.1137/110832513.

[32] R. U. Seydel. *Tools for Computational Finance*. Springer London, 2017.

[33] C. Thiery and B. Scherrer. Building controllers for Tetris. *Icga Journal*, 32(1):3–11, 2009. doi:10.3233/ICG-2009-32102.

[34] J. N. Tsitsiklis and B. van Roy. Optimal stopping of Markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. *IEEE Transactions on Automatic Control*, 44(10):1840–1851, 1999.

[35] W. R. Wade. *An Introduction to Analysis*. Prentice Hall/Pearson, 2010.

[36] White Goblin Games. The Game. `https://www.whitegoblingames.nl/game/277/The-Game`, 2019 (accessed June 21, 2019).

[37] M. Wiering and M. van Otterlo, editors. *Reinforcement Learning*. Springer Berlin Heidelberg, 2012.

Appendix

Recursive least squares method

Taken from [18]. Initialise θ_f^0 for all $f \in F$. Let B^0 be a $|F| \times |F|$ diagonal matrix with some small constant on the diagonal. Let $\phi(s_t^{x,n}$ denote the $|F|$ length vector containing the value of each feature in vector form. Then

$$B^n = \frac{1}{\alpha^n}\Big(B^{n\ -1} - \frac{1}{\gamma^n}(B^{n\ -1}\phi(s_t^{x,n})(\phi(s_t^{x,n}))^T B^{n\ -1})\Big), \tag{7.1}$$

and

$$\gamma^n = \alpha^n + \phi(s^{x,n}, t)^T B^{n\ -1}\phi(s_t^{x,n}). \tag{7.2}$$

Define

$$H^n = \frac{1}{\gamma^n} B^{n\ -1}. \tag{7.3}$$

Updating of θ_f^n is done by computing

$$\theta_f^n = \theta_f^{n\ -1} - H^n\phi(s_t^{x,n})(\overline{V}_t^{n\ -1}(s_t^{x,n}) - \hat{v}_t^n). \tag{7.4}$$

The parameter α^n determines the weight of the observations on the feature weights. We let

$$\alpha^n = 1 - \frac{\kappa}{n}, \tag{7.5}$$

where $\kappa \in [0, 1)$ is a parameter that requires initialising. Setting $\kappa = 0$ gives stationary data updating and other κ give non-stationary data updating.

$$SQ : \Big(C_t : C_t + 1, \max(h_{C_t}, h_{C_t+1}) + 1 : \max(h_{C_t}, h_{C_t+1}) + 2\Big)$$

$$T : \begin{cases} \Big(C_t, \max(h_{C_t\ \ 1}, h_{C_t} + 1, h_{C_t+1}\Big) \ \&\ \Big(C_t - 1 : C_t + 1, \max(h_{C_t\ \ 1}, h_{C_t} + 1, h_{C_t+1}) + 1\Big) & \text{if } \mathcal{R} = 0 \\ \Big(C_t - 1, \max(h_{C_t\ \ 1} + 1, h_{C_t} + 2)\Big) \ \&\ \Big(C_t, \max(h_{C_t} + 1, h_{C_t\ \ 1}) : \max(h_{C_t} + 1, h_{C_t\ \ 1}) + 2\Big) & \text{if } \mathcal{R} = 90 \\ \Big(C_t - 1 : C_t + 1, \max(h_{C_t\ \ 1}, h_{C_t}, h_{C_t+1}) + 1\Big) \ \&\ \Big(C_t, \max(h_{C_t\ \ 1}, h_{C_t}, h_{C_t+1}) + 2\Big) & \text{if } \mathcal{R} = 180 \\ \Big(C_t, \max(h_{C_t} + 1, h_{C_t+1})\Big) \ \&\ \Big(C_t + 1, \max(h_{C_t} + 2, h_{C_t+1} + 1)\Big) & \text{if } \mathcal{R} = 270 \end{cases}$$

$$I : \begin{cases} \Big(C_t, h_{C_t} + 1 : h_{C_t} + 4\Big) & \text{if } \mathcal{R} = 0 \\ \Big(C_t - 1 : C_t + 2, \max_{i2[C_t\ \ 1, C_t+2]}(h_i) + 1\Big) & \text{if } \mathcal{R} = 90 \end{cases}$$

$$RG : \begin{cases} \Big(C_t, \max(h_{C_t}, h_{C_t+1} + 1 : \max(h_{C_t}, h_{C_t+1} + 3)\Big) \ \&\ \Big(C_{t+1}, \max(h_{C_t}, h_{C_{t+1}}) + 1\Big) & \text{if } \mathcal{R} = 0 \\ \Big(C_t : C_t + 2, \max(h_{C_t} + 1, h_{C_t+1}, h_{C_t+2}) + 1\Big) \ \&\ \Big(C_t, \max(h_{C_t} + 1, h_{C_t+1}, h_{C_t+2})\Big) & \text{if } \mathcal{R} = 90 \\ \Big(C_t, \max(h_{C_t} + 1, h_{C_t\ \ 1} - 1) : \max(h_{C_t} + 3, h_{C_t\ \ 1} + 1)\Big) \ \&\ \Big(C_{t\ \ 1}, \max(h_{C_t\ \ 1} + 1, h_{C_t} + 3)\Big) & \text{if } \mathcal{R} = 180 \\ \Big(C_t : C_t - 2, \max(h_{C_t}, h_{C_t\ \ 1}, h_{C_t\ \ 2})\Big) \ \&\ \Big(C_t, \max(h_{C_t}, h_{C_t\ \ 1}, h_{C_t\ \ 2})\Big) & \text{if } \mathcal{R} = 270 \end{cases}$$

$$LG : \begin{cases} \Big(C_t, \max(h_{C_t}, h_{C_t\ \ 1}) + 1 : \max(h_{C_t}, h_{C_t\ \ 1}) + 3\Big) \ \&\ \Big(C_t - 1, \max(h_{C_t}, h_{C_t\ \ 1}) + 1\Big) & \text{if } \mathcal{R} = 0 \\ \Big(C_t : C_t + 2, \max(h_{C_t} : h_{C_t+2}) + 1\Big) \ \&\ \Big(C_t, \max(h_{C_t} : h_{C_t+2}) + 2\Big) & \text{if } \mathcal{R} = 90 \\ \Big(C_t, \max(h_{C_t} + 1, h_{C_t+1} - 1) : \max(h_{C_t} + 3, h_{C_t+1} + 1)\Big) \ \&\ \Big(C_t + 1, \max(h_{C_t} + 3, h_{C_t+1} + 1)\Big) & \text{if } \mathcal{R} = 180 \\ \Big(C_t - 2 : C_t, \max(h_{C_t\ \ 2} : h_{C_t}) + 1\Big) \ \&\ \Big(C_t, \max(h_{C_t\ \ 2} : h_{C\ \ t}) + 2\Big) & \text{if } \mathcal{R} = 270 \end{cases}$$

$$RS : \begin{cases} \Big(C_t, \max(h_{C_t} + 1, h_{C_t+1} + 2) : \max(h_{C_t} + 2, h_{C_t+1} + 3)\Big) \ \&\ \Big(C_t + 1, \max(h_{C_t}, h_{C_t+1} + 1) : \max(h_{C_t} + 1, h_{C_t+1} + 2)\Big) & \text{if } \mathcal{R} = 0 \\ \Big(C_t - 1 : C_t, \max(h_{C_t\ \ 1} + 1, h_{C_t} + 1, h_{C_t+1})\Big) \ \&\ \Big(C_t : C_t + 1, \max(h_{C_t\ \ 1} + 1, h_{C_t+1}, h_{C_t+1}) + 1\Big) & \text{if } \mathcal{R} = 90 \end{cases}$$

$$LS : \begin{cases} \Big(C_t, \max(h_{C_t} + 1, h_{C_t+1}) : \max(h_{C_t} + 1, h_{C_t+1}) + 1\Big) \ \&\ \Big(C_t + 1, \max(h_{C_t} + 2, h_{C_t+1} + 1) : \max(h_{C_t} + 3, h_{C_t+1} + 2)\Big) & \text{if } \mathcal{R} = 0 \\ \Big(C_t : C_t + 1, \max(h_{C_t} + 1, h_{C_t+1} + 1, h_{C_t\ \ 1})\Big) \ \&\ \Big(C_t - 1 : C_t, \max(h_{C_t} + 1, h_{C_t+1} + 1, h_{C_t\ \ 1}) + 1\Big) & \text{if } \mathcal{R} = 90 \end{cases}$$

Instances The Game

Table 7.1: Sequence of the numbers for each instance used for The Game.

Random 1			Random 2			Winnable 1			Winnable 2			Winnable 3		
99 (1st)	53	2	90	45	86	6	94	53	99	54	60	90	17	14
71 (2nd)	15	73	71	29	85	17	95	43	96	64	36	4	54	62
48 (3rd)	62	88	59	72	92	20	3	27	3	72	41	89	63	80
47	13	86	56	93	44	22	7	21	6	34	63	10	21	28
29	84	33	54	9	87	26	8	18	90	46	12	98	51	52
19	37	81	42	12	10	29	14	13	7	62	49	6	57	99
17	59	11	30	5	75	19	16	12	92	24	10	83	42	24
3	12	57	39	62	78	42	25	10	17	47	57	13	58	84
26	77	70	25	47	13	32	15	9	68	53	23	97	53	65
32	35	20	35	55	88	46	34	5	8	50	9	9	41	38
61	34	22	26	58	49	47	24	4	66	82	5	71	40	66
66	24	51	99	48	52	50	31	2	14	43	69	26	48	76
68	89	90	57	76	65	58	35	96	83	58	33	91	45	19
58	31	96	4	94	96	48	39	92	20	31	70	12	44	85
50	39	7	15	46	79	38	52	83	61	35	4	81	39	23
46	18	52	77	14	66	28	59	80	16	27	74	77	61	79
76	94	16	6	60	34	30	60	79	80	48	73	74	55	16
83	27	95	63	43	38	37	63	78	22	39	11	70	49	33
65	40	87	69	67	98	41	69	76	26	77	84	2	36	82
92	75	60	2	19	11	49	74	75	79	21	85	30	50	8
30	36	55	91	16	97	57	87	72	29	38	93	59	32	86
38	72	21	17	61	37	70	99	82	71	67	97	3	60	88
25	45	56	41	24	23	73	90	77	19	52	98	73	47	22
97	82	6	68	31	74	84	68	67	78	56	88	35	64	87
80	67	8	80	50	73	85	66	56	25	28	89	69	43	94
44	4	5	21	20	83	93	61	51	42	51	91	5	29	20
93	63	43	32	28	36	97	71	45	65	59	81	18	67	92
28	9	69	18	51	70	98	65	40	76	45	86	25	37	11
78	41	79	3	7	40	88	55	36	15	30	94	34	75	7
54	10	23	53	82	89	89	44	23	55	40	95	15	27	95
98	49	42	64	22	8	91	54	33	32	18	87	68	31	96
64	74	14	84	33	27	81	64	11	75	37	2	46	72	93
85	91		95	81		86	62		44	13		56	78	