University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Formal Modelling and Analysing Optimal Strategies for the Restless Multi-armed Bandit Problem

Archit Gupta

March 30, 2019

Abstract

Decision-making in noisy and changing environments requires a fine balance between exploiting knowledge about good courses of action and exploring the environment in order to improve upon this knowledge.

In this project we will look at the Restless Multi-armed bandit (RMAB) problem which are a class of sequential resource allocation problems concerned with allocating one or more resources among several alternative (competing) projects to model an agent to solve the exploration vs exploitation problem using Probabilistic model checking. RMAB is a challenging problem that is known to be PSPACE-hard in general.

Probabilistic model checking is a formal verification technique which is use to model and analyse stochastic systems such as these and to prove functional correctness of a system. The technique is used to study a wide range of quantitative properties of models like performance and reliability properties taken from many different application domains including computer and communication systems.

In this project, we give an overview of the probabilistic model checking tool PRISM, focusing in particular on models of discrete-time Markov chains and Markov reward models, and how these can be used to analyse performability properties. In particular, we use probabilistic model checking to analyze optimal strategies for the restless multi-armed bandit problem.

Acknowledgements

I would like to express my gratitude towards **Dr. Gethin Norman** for supervising this project and providing his invaluable advice and feedback throughout the academic year.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Archit Gupta Date: March 30, 2019

Contents

1 Introduction

This chapter provides the backdrop for this project. It not only defines the problem, but also explains why it is important, and the aim of this project.

1.1 The Problem

Mahajan and Teneketzis (2008) describes the multi-armed bandit problem (MAB) in probability theory, as a classical reinforcement learning problem that illustrates the exploration-exploitation trade-off dilemma. In the problem, we try to maximize our gain when a fixed limited set of resources are to be allocated between competing (alternative) choices, when knowledge about choices are only partially known at the time of allocation, and may become better understood by allocating resources as time passes. The name comes from imagining a gambler at a row of slot machines, who has to decide which machines to play, how many times to play each machine and in which order to play them, and whether to continue with the current machine or try a different machine. The problem also falls into the broad category of stochastic scheduling.

According to Gittins et al. (2011), in the MAB problem, we try to model a realistic agent that tries to balance the exploration-exploitation trade-off, in order to maximize their total value over the period of time considered. The agent tries to acquire new knowledge about the system at every state known as exploration while also attempting to optimize their decisions based on existing knowledge gained from the system known as exploitation. The problem requires balancing reward maximization based on the knowledge already acquired with attempting new actions to further increase knowledge.

The multi-armed bandit model has many variations including the Binary multi-armed bandit or Bernoulli multi-armed bandit, which issues a reward of one with probability p, and otherwise a reward of zero. In this project we will focus on a extended formulation of the MAB known as restless multi-armed bandit (RMAB) which has each arm representing an independent Markov machine (discussed in Chapter 3) (Dai et al. 2011). An arm is a representation of an action chosen by the agent. Each time a particular arm is played, the state of that machine advances to a new one, chosen according to the Markov state evolution probabilities. There is a reward depending on the current state of the machine. In the RMAB variation, the states of non-played arms also evolve over time.

1.2 Motivation

In the RMAB problem, each machine provides a random reward from a probability distribution specific to that machine. The objective of the gambler is to maximize the accumulated rewards earned through a sequence of actions performed by it. The crucial trade-off the gambler faces at each trial is between exploration to get more information about the expected rewards of the other machines and exploitation of the machine that has the highest expected reward. The trade-off between exploration and exploitation is also faced in machine learning. In practice,

multi-armed bandits have been used to model problems such as managing research projects in a large organization like a science foundation or a pharmaceutical company. This is in order to allocate fixed budgets to a variety of research projects that show a high degree of uncertainty at the start, but a clearer outcome (or lack of it) as they progress.

According to Whittle (1988), when designing systems (hardware or software) it is important to know whether or not your system will operate as expected/required. For example we do not want to find out that our nuclear power plant control system after implementation is not as safe as was expected. There are several techniques available that can be used to verify the functional correctness of a system. Examples of such techniques are: theorem proving, simulation/testing and model checking (Oldenkamp 2007). This project focuses on the last technique, namely formal verification by means of model checking. The goal of this technique is to predict system behaviour, or more specifically, to formally prove that all possible executions of the system conform to the requirements.

1.3 Aims

Automated verification techniques such as probabilistic model checking used in this project are useful to analyse quantitative properties of these systems and to build confidence in their functional correctness. The objective of this project is to model the restless multi-armed bandit problem using Discrete Time Markov Chains (DTMCs), Markov Decision Process (MDPs) and Partially Observable MDPs (POMDPs). There will be experiments using certain hypothesis supported by generated strategies based on models with different numbers of bandits with various different probabilities. To achieve this we will use PRISM (Kwiatkowska et al. 2011), a probabilistic model checking tool to model and analyse our RMAB system.

1.4 Outline

In this project, we look at the formal modelling and analysis of the optimal strategies of the RMAB problem. This included studying about the problem and models first and then modelling them in PRISM. I will also present the tools used to automate and build the strategy graphs. The document is divided into the following chapters:

Chapter 2 - **Background and Application** This chapter discusses background on the problem and some real world applications.

Chapter 3 - **Probabilistic Model Checking** This chapter discusses the background knowledge needed on probabilistic model checking.

Chapter 4 - **Probabilistic Models** This chapter discusses the three models (DTMC, MDP and POMDP) and their formal modelling in PRISM in detail .

Chapter 5 - **Analysis and Tools** This chapter discusses the tools developed and the analysis of the optimal strategies.

Chapter 6 - **Conclusion and Future Work** This chapter summarises the project report and suggests some ideas for future work.

2 Background

In this chapter we discuss the RMAB problem in detail by explaining multi-armed bandits, extending them to restless multi-armed bandits and also provide some real world applications.

2.1 Multi-armed Bandit

The multi-armed bandit problem is a classic version of the problem of optimal allocation of activity under certainty (Akin 2017). From the algorithmic point of view, consider an agent that has no knowledge of the environment and can only learn by trial and error from a single reward signal which notifies the agent whether it failed or succeeded. Imagine a player in a casino, purchasing n coins with which they want to play on slot machines called “Multi-Armed Bandit" (MAB) as each slot machine’s lever or handle represents an “Arm". Further, we assume that each slot machine or arm can give a reward based on a probability distribution for each on and each arm’s reward is a realization or sample from that distribution. It is assumed the player, does not initially know which arm will give the best payout (i.e. has the highest chance of reward), and therefore they must use a few coins to explore the system, and compare the arms. Now, the player has rough estimates on the potential rewards from the arms and must decide to either keep exploiting the assumed arm to yield the best reward so far found or continue to explore the rest of the arms with the finite coins. This tension between exploration and exploitation is the heart of the MAB Family of Problems.

We can also think about the problem as a class of sequential resource allocation problems by imagining a player with a single resource which they have to allocate to one project at each finite time step among a number of alternative or competing projects. When the allocation of the resource is done, that project changes its state while the rest stay static. Further, the rewards or the return on investments from each project can be different. Therefore, Most MAB algorithms use the player’s regret as their measure of effectiveness. The regret of a system is the difference between the arm they chose and the “best" arm that the they could have chosen at that time, had they known all of the probability distributions.

There are of course many variants of this problem or ways to adjust the classic setup to fit numerous real-world situations such as: one or multiple resources available for allocation, new projects appearing over time, all projects changing each time step, or even a strategy which chooses the rewards of each arm. Consider a casino with K slot machines, each of these "arms" denoted by $i \; 2 \; K$. At each discrete time step, $n = 1; 2; 3; : : :$; the player must decide which of the K arms to pull. Each arm, i, returning a random, positive, real-valued, reward in the range of »0; 1¼. The rewards returned from each arm, immediately after pulling that arm, are independent and identically distributed as well as independent of the play of the other arms. Therefore, the player or MAB algorithm must decide which arm to pull, at time n, based on the rewards received i.e the knowledge gathered up through time $n \quad 1$. The goal therefore, is to maximize the total expected reward by time K.

2.2 Restless Multi-armed Bandit

In this project we focus on the above casino problem mentioned above, modeled as an RMAB, a generalization of the classic MAB problem. In the problem there are several arms, each of which is modelled as a Markov chain. In contrast to the above MAB model, in RMAB, the state of each arm continues to evolve even when it is not played. More specifically from Liu et al. (2013), each arm changes its state according to an unknown Markovian transition rule when the arm is played or not played. An arm can have multiple states which represents different values they are assigned with, for example a coin has two states, namely heads and tails. The states of the arms are unknown, and at each time step an arm is played. The played arm reveals its state and yields a reward, which is a function of the state. The objective is to find a policy that maximises the total reward over some period of time. RMAB problems have been shown to be, in general, PSPACE hard (Papadimitriou and Tsitsiklis 1994), and our knowledge on the structure of the optimal policy for general RMAB problem is limited.

2.3 Applications

We now look at three Real World applications where a problem is formulated into an RMAB problem.

Example 1. Wireless communication devices can often operate on several alternative channels. For example, phones are equipped with base-station communication capability along with WiFi and Bluetooth communication may use several frequency bands (Kuhn and Nazarathy 2017). To maximize the long-run average throughput which is the amount of items that can be transferred at a given time, automatic decision support systems in such devices need to decide which channels to use. An optimal decision policy needs to take into account that, due to time and resource constraints, the state of a channel is only observed when it is selected for transmission. Therefore, with respect to long-run average throughput, the greedy strategy of just exploiting the channels assumed to yield the currently highest transmission rate is not necessarily optimal and rather, it may be favourable to give more priority to the exploration of channels of uncertain quality. Therefore, it is crucial to make the trade-off.

At first, the channels operate as independent Markov processes and the transmitter has the choice to use from the available channels at every discrete time step and they cannot be used in parallel due to hardware limitations, resource availability and/or energy constraints. We assume that the models and their parameters are known, and the channel we pick, yields an immediate reward that depends on the state of the channel which can be for example, the number of bits it can transfer successfully. In an ideal situation the transmitter knows the states of all channels, and therefore will choose the channels in a way that achieves the largest throughput over time. However, the nature of communication channels is often stochastic and due to resource constrains in the real world, the transmitter does not know the current state of each channel with certainty and the state of a channel is only observed when selected. The unselected channels on the other hand are not observed in this time instance. This type of problem is a special case of a RMAB.

Example 2. From Blasco and Gündüz (2015) we know, low power wireless networks, such as machine-to-machine and wireless sensor networks, can be complemented with energy harvesting (EH) technology to extend the network lifetime. Due to the battery size, a low-power wireless node has a limited lifetime but when an EH device and a rechargeable battery can be added, its lifetime can be prolonged significantly. However, due to the random nature of the energy sources

where the energy in arbitrary amounts arrives at random times and energy availability being scarce at the EH nodes, to make the most out of the scarce energy, it is important to optimise the scheduling policy of the wireless network by studying an online scheduling of low-power wireless nodes by an access point (AP).

The nodes are powered by rechargeable batteries and equipped with EH devices and at each time step (TS) a node which is dependent on the channel conditions or the availability of data at the node is operative with a certain probability distribution. The EH process at each node is modelled as an independent Markov process, and at each TS, a node will either harvest one unit of energy or will not harvest any. At each TS, the AP is in charge of scheduling and the EH attaches to the available orthogonal channels. A node transmits only when it is operative and is scheduled at the same time. Hence, at each TS the AP learns the EH process states and battery levels of the operative nodes that are scheduled, but does not receive any information about the other nodes. The AP is interested in maximising the expected accumulated throughput within a given time frame. Therefore, this problem is modelled into an RMAB problem.

Example 3. Unmanned aerial vehicles (UAVs) are already actively being used in military operations and are being investigated for civilian applications such as environmental control and monitoring. Although there are technological advancements made everyday, it seems clear that a major challenge for future developments will be to increase the level of autonomy of these systems (Le Ny et al. 2008). Lets consider a UAV with M mobile sensors which are tracking N sites where $N > M$ and each site can be in one of two states (s_1 or s_2) at each period of time. The system observes the state of the site if visited by a sensor. For example, marine vessels have bilges where excess water and oily wastes can collect, therefore the state here represents if the ship's bilge is dumping excess which is state s_1 and state s_2 if otherwise. However, we also know that the states of the sites evolve independently of each other and and as Markov chains with known probabilities and therefore, at a given time period, we can estimate the states of the sites that are not visited by any sensor. Now, lets take that whenever a sensor visit a site which happens to be in state s_1, we receive a reward and if the site is in state s_2, we do not obtain a reward. The goal is to allocate the sensors at each time period, in order to maximize the expected total discounted cost over a given time frame. Through this model, we capture the following trade-offs. It is advantageous to explore and keep a good knowledge of the states of the sites in order to take an optimal decision about which sites to visit next. However, the estimation problem is not the end goal, and so there is a balance between visiting sites to gain more information and collecting rewards. The rewards are associated with the detection of a dumping event (state s_1 for a ship). However, if the UAV is not present during the dumping, the event is missed.

2.4 Chapter Summary

In this chapter, the MAB problem was discussed and the extension of the problem we look at which is the RMAB problem where there are several arms and each arm has a potential for a reward and also evolves independently of the other arms and also when they are not played and how we need to balance exploring the system using resources and exploit the system for the best reward. It was also shown how an allocation under uncertainty and the exploration-exploitation trade off problem relates to the real world problems.

On the basis of the formulation of the problem, the next chapter identifies the technique and tool we use to model and analyse the problem.

3 Probabilistic Model Checking

In this chapter, we introduce model checking and probabilistic model checking, then the modelling language and tool PRISM which was used in this project to develop and analyse the models.

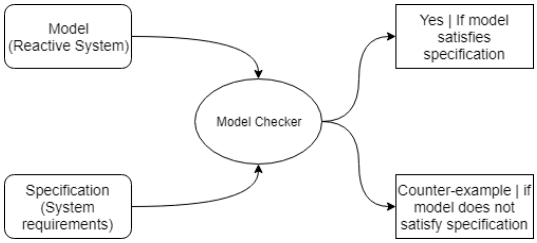
3.1 Model Checking

Clarke et al. (1999) defines model checking as a method for formally verifying and proving functional correctness of finite-state concurrent or parallel systems. Given a model of a system, model checking exhaustively and automatically checks whether the model meets a given specification. Specifications about the system are expressed as temporal logic formulas, and efficient symbolic algorithms check if the specification holds or not by traversing the models and extremely large state-spaces can often be traversed in minutes. According to Clarke and Emerson (1982) both the model of the system and the specification are formulated in some precise mathematical language in order to solve such a problem algorithmically. The problem is formulated as a task in logic, namely to check whether a given structure satisfies a given logical formula. This general concept applies to many kinds of logics and suitable structures. A simple model checking problem is verifying whether a given formula in the propositional logic is satisfied by a given structure. The essential idea behind model checking is shown in Figure 3.1. A model-checking tool accepts system requirements or design (called model) and a property (called specification) that the final system is expected to satisfy. The tool then outputs yes if the given model satisfies given specification and generates a counterexample otherwise where it details why the model does not satisfy the specification. By studying the counterexample, you can pinpoint the source of the error in the model, correct the model, and try again. The idea is that by ensuring that the model satisfies enough system properties, we increase our confidence in the correctness of the model. A logical approach leads to unambiguous specifications and model checking leads to automatic reasoning.

As the name suggests, model checking is performed on a model of a system (Oldenkamp 2007). The model is usually generated from a high level system description language, such as process algebra or Petri net. Typically, these generated models are non-deterministic finite-state automata which are transition systems that describe the system's possible behaviours. They can be considered as directed graphs consisting of a finite set of states (vertices) labelled with atomic propositions and state transitions (edges) that show how the system can change from one state to another. The atomic propositions represent the basic properties that hold in each state. Once the system is represented by such a model we can check if the model satisfies a formal specification (i.e. if it has certain properties). From Clarke and Emerson (1982), the properties that are checked against the system model are expressed using temporal logics, such as LTL (Linear Time Logic) or CTL (Computation Tree Logic). For example, CTL can specify that when some initial condition is satisfied (e.g., all program variables are positive or no cars on a highway straddle two lanes), then all possible executions of a program avoid some undesirable condition (e.g., dividing a number by zero or two cars colliding on a highway). These logics can express properties on states or paths in the automata. Once a model and property have been formulated the model checking

process will perform a systematic state-space exploration to verify if the property holds. This form of traditional model checking focuses on delivering a 100% accurate guarantee whether or not a property holds. There are many cases where giving an absolute guarantee is not possible. Examples are communication protocols, such as Bluetooth or IEEE 802.11 Wireless LAN, that have to deal with a certain probability of message loss. This is where probabilistic model checking can be utilized.

Figure 3.1: *Model Checking Approach*



3.2 Probabilistic Model Checking

According to Oldenkamp (2007) probabilistic model checking is an automatic formal verification technique for the analysis of systems which exhibit stochastic behaviour i.e exhibit probabilistic behaviour. The technique is similar to model checking as discussed earlier. The major difference is that a probabilistic model contains additional information on the likelihood or timing of transitions between states, or in simpler terms, it can model stochastic behaviour. Probabilistic model checking refers to a range of automated techniques for calculating the probability of the occurrence of certain events during the execution of the system, and can be useful to establish properties such as “shutdown occurs with probability at most 0.01" and “the video frame will be delivered within 5ms with probability at least 0.97". Models can also be augmented with Costs and Rewards which can be properties like the reward accumulated along a path until a certain state has been reached known as “Reachability reward" or “Cumulative reward" properties which for a given time bound associate a reward with each path of a model, for example, “the expected time taken to reach from a state s to a state where z equals 2" or “the expected number of lost requests in 15.5 time steps". Applications range from areas such as randomized distributed algorithms to planning and AI, security, and even biological process modelling or quantum computing.

A probabilistic model checker also has a similar behaviour shown in Figure 3.1 where it takes as input a probabilistic property and a probabilistic model and delivers the result "Yes" or "No" (i. e. whether or not the property is satisfied) or quantitative results. From these inputs, a model checker can construct a model of the system, which is a labelled state-transition system where each state represents a possible combination or arrangement and each transition represents an evolution over time of the system from one state to another. It is then possible to automatically verify whether or not each property is satisfied, based on a systematic and exhaustive exploration of the constructed state-transition system. We know from Forejt et al. (2012), in probabilistic model checking, the models are augmented with quantitative information regarding the likelihood that transitions occur and the times at which they do so. Model checking reduces to an optimisation problem, namely determining the maximum (or minimum) probability (or expected cost/reward) achievable by the model and the reward operator enables the computation of expectations with respect to the underlying probability space. In practice, these models are typically Markov chains or Markov decision processes. The following sections elaborate on the on the tool we use for this project.

3.3 PRISM

Kwiatkowska et al. (2011) defines a probabilistic model checker tool can verify if a system - which is described by a model, written in a formal language - satisfies a formal specification, which is expressed using logics, such as Probabilistic Computation Tree Logic (PCTL). In simple terms, the purpose of the PRISM tool is to construct a probabilistic model, from high-level descriptions, and then analyse one or more properties. PRISM is a probabilistic symbolic model checker, developed at the University of Birmingham (United Kingdom) for the modelling and analysis of probabilistic systems. It has been used to analyse systems from many different application domains, including communication and multimedia protocols, randomised distributed algorithms, security protocols, biological systems and many others.

From Kwiatkowska et al. (2010), system models are described using the PRISM programming language, which is a high-level state-based description language. In this language a system is described as the parallel composition of a set of modules. A module state is determined by a set of finite-range variables and its behaviour is given using a guarded-command based notation. Communication between modules takes place either via global variables or synchronisation over common action labels. Besides its own model description language, PRISM supports a subset of PEPA, which is a stochastic process algebra. PRISM also provides indirect support for model checking probabilistic timed automata which include probability, non-determinism and real-time using clocks. In PRISM it is possible to either determine if a probability satisfies a given bound or obtain the actual value. There is also support for the specification and analysis of properties based on costs and rewards.

3.4 Chapter Summary

In this chapter, we discussed the probabilistic model checking technique which given a model and its requirements, automatically verifies and proves the functional correctness of the model and can either simply give a “yes" or “no" answer or some quantitative results. We also introduced the model checking tool PRISM which we used in this project to model and analyse the problem.

On the basis of the background of model checking and the problem, the next chapter discusses the three models and how to model them in PRISM.

4 Probabilistic models

In order to model check a system that exhibits stochastic behaviour we will first need to build a formal probabilistic model of that system. There are several commonly used probabilistic model representations for stochastic system, for example:

- Discrete-Time Markov Chains (DTMC)
- Continuous-Time Markov Chains (CTMC)
- Markov Decision Process (MDP)
- probabilistic automata (PAs)
- probabilistic timed automata (PTAs)

From Oldenkamp (2007) we know, that these models usually form a combination of probability theory and graph theory. They consist of states, transitions, and arcs that connect them. From now on we will focus on discrete time models. These particular models are used by the PRISM model checker tools studied in this thesis. Such models should be considered as a transition system, where we can move from one state to another and the choice of which state to go to depends on some probability distribution. Moving from one state to another is referred to as "making a transition". A more formal definition will be presented later on in this chapter. We consider three different types of models, which are supported by PRISM, and use them to model our RMAB problem while providing the PRISM version of the models used in this project namely:

1. DTMC - Discrete-Time Markov Chain
2. MDP - Markove Decision Process
3. POMDP - Partially Observable Markov Decision Process.

4.1 Discrete Time Markov Chains

Sigman (2008) defines a stochastic process in discrete time $n \in \mathbb{N} = \{0; 1; 2; \ldots\}$ as a sequence of random variables $X_0; X_1; X_2, \ldots$ denoted by $X = \{X_n \mid n \quad 0\}$ (or just $X = \{X_n\}$). We refer to the value X_n as the state of the process at time n, with X_0 denoting the initial state. If the random variables take values in a discrete space such as the integers $\mathbb{Z} = \{\ldots; 2; 1; 0; 1; 2; \ldots\}$ (or some subset of them), then the stochastic process is said to be discrete-valued.

Stochastic processes are meant to model the evolution over time of real phenomena for which randomness is inherent. For example, X_n could denote the price of a stock n days from now, the population size of a given species after n years, the amount of bandwidth in use in a telecommunications network after n hours of operation, or the amount of money that an insurance risk company has right after it pays out its n^{th} claim. The insurance risk example illustrates how "time" n need not really be time, but instead can be a sequential indexing of some kind of events. Other such examples: X_n denotes the amount of water in a reservoir after the n^{th} rain storm of the year, the amount of time that the n^{th} arrival to a hospital must wait before being admitted, the outcome (heads or tails) of the n^{th} flip of a coin.

A discrete time Markov Chain is a sequence of random variables $X_0; X_1; : : :$ indexed by time taking values in some state space, with the property that a future state X_{t+1} is only dependent on the current state $X_t = x$, and therefore conditionally independent of the past. We call this independence the **Markov property**. While this may seem like a rather large assumption to make, if needed, we can embed information about the past into the current state thereby maintaining this assumption without losing the mathematical power of the memoryless property of Markov Chains.

According to Oldenkamp (2007) DTMCs are stochastic models of systems with countable state-space that change their states at times $n = 0; 1; 2; :::$ and have the following property: if the system enters state s at time n, it stays there for exactly one unit of time and then jumps to state s^0 at time $n + 1$ with probability $P(s; s^0)$, regardless of its history up to and including time $n - 1$. Formally we have the following definition Kwiatkowska (2003).

Definition 1 *A DTMC is a tuple* $D = (S; \bar{s}; P; AP; L; R)$ *where:*

- S *is a set of state;*
- $\bar{s} \in S$ *is in initial state;*
- $P : S \times S \to [0, 1]$ *is a probabilitiy matrix such that* $\sum_{s' \in S} P(s; s') = 1$*;*
- AP *is a set of atomic propositions;*
- $L : S \to 2^{AP}$ *is a labelling with atomic propositions that are true in state.*
- $R = (R_S; R_T)$ *is a reward structure where* $R_S : S \to \mathbb{R}$ *is a state reward function and* $R_T : S \times S \to \mathbb{R}$ *a transition reward function.*

The definition shows that states are labelled with atomic propositions, they indicate for instance the status of the system (e.g. being on and off). The system can change state according to a probability distribution given by the transition probability matrix. A transition from state s to s^0 can only take place if $P(s; s') > 0$. If $P(s; s') = 0$, no such transition is possible. The system can occupy the same state before and after the transition. A sequence of states and transitions forms a **path** π, where a path is defined as a finite or infinite sequence $s_0 \xrightarrow{p_0} s_1 \xrightarrow{p_1} \cdots \xrightarrow{p_i} s_i \cdots$ with $i \in \mathbb{N}; s_i \in S$, and $p_{i_{[0,1]}}$ such that $p_i > 0 \wedge p_i = P(s_i; s_{i+1})$ for all $i \geq 0$ (Oldenkamp 2007). We can define a probabilistic measure $Prob^{\pi}(D)$ over the set of paths $Path^{\pi}(D)$.

DTMCs (and other probabilistic models) can also be augmented with reward information, which allows reasoning about a wide range of additional quantitative measures. Formally from Kwiatkowska et al. (2007), for a DTMC D, we included a reward structure $R = (R_S; R_T)$ that allows specification of two distinct types of rewards mentioned. The state reward R_S is the reward acquired in state s per time step, i.e. a reward of $r(s)$ is incurred if the DTMC is in state s for 1 time step and the transition reward R_T is acquired each time a transition between states s and s^0 occurs. Reward structures can be used to represent a variety of different aspects of a system model, for example "the number of messages successfully transmitted by a protocol" or "the amount of time that a server spends operational". Note that state rewards are sometimes called cumulative rewards while transition rewards are sometimes referred to as instantaneous or impulse rewards.

A DTMC admits probabilistic choice only, which substantially differs from non-determinism: the frequency of a probabilistic edge being taken is determined by the distribution, whereas nondeterministic choice is made by the environment which has complete freedom to select any of the alternatives. Note also that there is no notion of real time, though reasoning about discrete time is possible through state variables keeping track of time and 'counting' transition steps. Below is the qualitative measure which together form the basis for probabilistic model checking of DTMCs. We will also extend this to our MDP and POMDP models.

Expected Reachability (Kwiatkowska et al. 2006) The measure we consider is expected reachability, denoted by $\mathbb{E}$ (Expectation) which refers to the expected reward accumulated, starting in state s, before reaching a set $F \subseteq S$ of target states.

We define Expected Reachability for a path π, by

$$E_D^{\pi}[F] \stackrel{def}{=} \int_{\pi \in Path^{\pi}[D]} r[F;\pi]\, dProb_D^{\pi}$$

where $r[F;\pi]$ is the total reward accumulated until a set of states state F is reached along for any infinite path π.

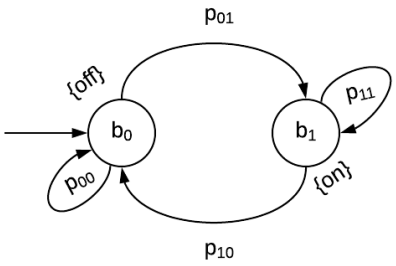
In PRISM properties such as expected reachability are defined using the property specifications which are used to analyse the models. Properties of PRISM models are expressed in PCTL. The operators $P_{\bowtie p}$, $S_{\bowtie p}$ and $R_{\bowtie r}$ by default include the probability bound p or reward bound r. However, in PRISM, we can also directly specify properties which evaluate to a numerical value by replacing the bounds in the P, S and R operators with =?, as illustrated in the following PRISM specification examples:

- $P_{max=?}[F <= T messages\_lost > 10]$ – is the maximum probability of messages greater than 10 have been lost by time T
- $S_{=?}[queue\_size > max\_size \cdot 0.75]$ – the long-run probability where a queue's size is more than 75% full
- $R_{=?}[S]$ – the long-run expected queue-size.

The meaning ascribed to these properties are dependent on the definitions of the atomic propositions and reward structures. The property we will use to analyse our models is:

- $R_{=?}[F \text{ "prop"}]$ – corresponds to the expected reward cumulated along a path until a state satisfying property "prop" is reached.

Figure 4.1: *A bandit state transition*



Example 1. We represent a DTMC as a transition diagram, Figure 4.1 shows a simple example of a DTMC $D = [B; \bar{b}; P; AP; L]$ where states are drawn as circles and transitions as arrows, labelled with their associated probabilities. The initial state is indicated by an additional incoming arrow. The DTMC D has two states: $B = \{b_0; b_1\}$, with initial state $\bar{b} = b_0$. The transition probabilities are defined as $p_{i,j}$ which is the probability of transition from state i to j. The atomic propositions used to label states are taken from the set $AP = \{off; on\}$. Here, the DTMC models a simple process where a bandit changes its state based on a probability distribution. After one time-step, it enters the state b_0 from which, with probability p_{00} it waits another time-step, with probability p_{01} it changes its state from 0 to 1, and with probability p_{10} it changes its state back. In the latter case, the process restarts. There is also a reward of 1 associated whenever the bandit is in state b_1. The labelling function allows us to assign meaningful names to states of the DTMC:

$$L_1[b_0] = \{off\};\ L_1[b_1] = \{on\}:$$

Model 1 Example DTMC

```
dtmc //the model type

const double p01 = 0.3 //transition probability from 0 to 1
const double p10 = 0.4

module bandit1
   b1: [0..1]; //variable

   [] b1=0 -> p01:(b1'=1) + (1-p01):(b1'=0); //when b1=0. p01 is the
       probability for becoming b1=1
   [] b1=1 -> p10:(b1'=0) + (1-p10):(b1'=1);

endmodule

rewards "correct_guess" //name for the reward
   b1=1 : 1; //when b1=1, give a reward of 1
endrewards
```

We now briefly introduce the PRISM modelling language and explain how to model our Example in PRISM. According to Kwiatkowska et al. (2009) a PRISM model comprises of the type of probabilistic model and a set of modules which represent different components of the system being modelled. As mentioned previously, PRISM can be used to describe several types of probabilistic models by including their keywords: dtmc, ctmc, mdp, pta or pomdp which is typically specified at the start of the file. If no model type is declared, then the model is by default assumed to be an MDP. The state of each module is modelled a variable representing a set of finite-randing values. The global state of the model at any point in time is determined by the values of all such module variables and, optionally, a set of global variables. The behaviour of a module, i.e. the changes to its state that can occur, is specified by a set of guarded commands. These take the form:

$$\text{»act¼ guard ! rate : update;}$$

where `act` is an (optional) action label, `guard` is a predicate over the variables of the model, `rate` is a positive real-valued expression and `update` is of the form:

$$^1x_1^0 = u_1{}^o \;\&\; ^1x_2^0 = u_2{}^o \;\&\; \cdots \;\&\; ^1x_k^0 = u_k{}^o$$

where $x_1; x_2; \ldots; x_k$ are local variables of the module and $u_1; u_2; \ldots; u_k$ are expressions over all variables. If the guard is true, a transition which the module can make is descibed by each update. From Kwiatkowska et al. (2007), by giving the new values of the variables in the module, a transition is specified and possibly as a function of other variables. In general, a PRISM language description is the parallel composition of a probabilistic model modules. In every state of the model, there is a set of commands whose guards are satisfied in that state. Each update is also assigned a probability which will be assigned to the corresponding transition. Finally, to specify additional quantitative measures of interest, we discuss the addition of rewards to a model. Reward structures are associated with models using the **rewards** “*rewardname*”:::*endrewards* construct and are specified using multiple reward items of the form: *guard* : *rate*;*or»act¼guard* : *rate*; depending on whether a state or transition rewards are being specified. A single reward item depending on the values of model variables in each one, can assign different rewards to different states or transitions and any states or transitions which do not satisfy the guard of a reward item

will have no reward assigned to them. The sum of the rewards is assigned for all the corresponding reward items for the states and transitions which satisfy multiple guards.

Model 1 shows the PRISM modelling language version for Example 1, where first we specify that our model is a DTMC by adding the keyword dtmc. A module bandit1 is now described having a variable $b1$ which is assigned to values ranging from 0 to 1, describing its off and on state, and the first command, for example, describes the probabilities of the bandit being either staying off or transitioning its state to on. The bandit changes its state depending on the probabilities in every round. A reward of one is also given using the reward structure whenever the bandit is on (i.e $b1$ is equal to the value 1).

We now model our RMAB problem as a DTMC using PRISM as shown in Model 2 by extending our *Example 1*. The model includes a set of bandits with varying transition probabilities, and a scheduler which picks which bandit to choose. Synchronization is used such that both sets of bandits change their states at the same time by labelling commands with actions that are placed between the square brackets. As can be seen from the comments labelling the code, there are two state of either bandit, which are simply an integer value (between 0 and 1) representing both bandits in the off or on state. In every state s of the model, there is a set of commands (belonging to any of the modules) which are enabled, i.e. whose guards are satisfied in that state. The state of the scheduler s is an integer value (between 0 and n) where state 0 is when the bandit state are changing and n being the number of bandits, which also corresponds where state number k 2 n is the bandit the system chooses, for example, $s = 1$ is when the system chooses bandit 1. Turn is used to represent rounds for the system where the system will first change the states of bandits then choose a bandit. Turn 1 to 0 is only used to initialise bandits where they have a 50% chance of either being "on" or "off". Turn 0 to 1 represents the synchronised movement of every bandit changing their state at the same time, and turn 1 to 0 represents the system picking a bandit. These two transitions form a round of the system and is repeated until our counter reaches the value *kmax*. *kmax* is a constant which forms the module counter in which a global variable k is an integer value ranging from 0 to *kmax* + 1. The counter provided is used for our property specification $R_{=?}$ » $F\ k = kmax + 1$ ¼ which is for computing the expected reward cumulated along a path for n number of iterations. The choice between which command is performed (i.e. the scheduling) depends on the model type and since our model is a DTMC, the choice is probabilistic, with each enabled command selected with our choice of probability. The probability matrices of $b1$ and $b2$ is as follows:

$$P_1 = \begin{matrix} 0{:}7 & 0{:}3 \\ 0{:}6 & 0{:}4 \end{matrix} \quad \& \quad P_2 = \begin{matrix} 0{:}5 & 0{:}5 \\ 0{:}35 & 0{:}65 \end{matrix}$$

The behaviour of our model is as follows. Starting in the initial state s_0, the scheduler performs *initial* action which is used to initialize our bandits to a random state initially so they have 50% of being either 0 or 1 (turn –1 to 0). The scheduler now performs action $a1$ (turn 0 to 1) which changes the states of all the bandits simultaneously. After that, *choose* (turn 1 to 0) is performed where as the model being a DTMC, the scheduler picks one bandit randomly, i.e the probability of choosing either bandit is 0.5, hence the model does not learn or predict anything based on current/previous states. The rewards module here associate a state reward of 1 to the state in which the system picks either bandit 1 or bandit 2 when they are in the "on" (value 1) state by using the two reward structures. Afterwards, the scheduler repeats the process from rotating the bandits to picking them, and so forth.

Model 2 RMAB Problem in DTMC

```
dtmc //model type

const double p1 = 0.3; //bandit 1 going from 0 to 1
const double p2 = 0.4; //bandit 1 going from 1 to 0
const double p3 = 0.5; //bandit 2 going from 0 to 1
const double p4 = 0.35; //bandit 2 going from 1 to 0

module bandit1
   b1: [0..1];  // bandit is either off or on
   [initial] true -> 0.5 : (b1'=0) + 0.5 : (b1'=1); //initially 50-50
       chance of b1=0 or b1=1.

   [a1] b1=0 -> p1:(b1'=1) + (1-p1):(b1'=0); //defining probs at which
       b1=0 can stay or transition to b1=1
   [a1] b1=1 -> p2:(b1'=0) + (1-p2):(b1'=1);

endmodule

module bandit2 //same as above with different probabilities

     b2 : [0..1];
     [initial] true -> 0.5 : (b2'=0) + 0.5 : (b2'=1);

     [a1] b2=0 -> p3:(b2'=1) + (1-p3):(b2'=0);
     [a1] b2=1 -> p4:(b2'=0) + (1-p4):(b2'=1);
endmodule

module scheduler
     s : [0..2]; // 0 to n where n is the number of bandits.
     //s=0 is no bandit chosen
     turn : [-1..1]; //specifying if scheduler is picking or rotating
         the bandits
     [initial] turn=-1 -> (turn'=0); //turn = -1 is used to just
         initialise the bandits randomly

     [a1] turn=0 -> (turn'=1); //this turns all the bandits
     [choose] turn=1 -> 0.5:(s'=1)&(turn'=0) + 0.5:(s'=2)&(turn'=0);
         //picks bandit randomly

endmodule

//running the property for kmax iterations
const int kmax; //used in counter

module counter

  k : [0..kmax+1];
  [a1] k<kmax -> true; //turn bandits until kmax is reached
  [choose] k<kmax -> (k'=min(kmax,k+1)); //choose bandits until kmax
      is reached
  [] k=kmax -> (k'=k+1); //make the last iteration

endmodule

rewards "correct_guess"
  turn=0 & s=1 & b1=1 : 1; //give a reward of 1 when bandit 1 is
      picked when it is on
  turn=0 & s=2 & b2=1 : 1;
endrewards
```

4.2 Markov Decision Process

We can now extend our DTMC of the RMAB as an Markov decision processes (MDPs) to allow the scheduler to choose which arm to play as opposed to the random choices and where the state of every bandit is fully observable. We work with MDPs, a widely widely used model for systems which exhibit both non-determinism, due for example to concurrency, and probability, representing for example randomisation or unpredictability. An MDP is an extension of DTMC where instead of making a probabilistic choice, MDPs make a nondeterministic choice between several discrete probability distributions over successor states. From Norman et al. (2017), let $Dist(X)$ denote the set of discrete probability distributions over a set X and $\mathbb{R}$ used earlier.

Definition 2 *MDP An MDP is a tuple* $M = (S, \bar{s}, A, P, R)$ *where:*

- S *is a set of states;*
- $\bar{s} \in S$ *is an initial state;*
- A *is a set of actions;*
- $P : S \times A \to Dist(S)$ *is a (partial) probabilistic transition function;*
- $R = (R_S, R_A)$ *is a reward structure where* $R_S : S \to \mathbb{R}$ *is a state reward function and* $R_A : S \times A \to \mathbb{R}$ *an action reward function.*

This represents an MDP M which can model a system exhibiting both probabilistic and non-deterministic behaviour through states from the set S. Each state s of M has a set of available actions represented by the set of $A(s) \to \{a \in A \mid P(s,a) \text{ is defined}\}$ where each available action/transition is chosen non deterministically.

A path of an MDP M represents a particular resolution of both non-determinism and probability. Formally from Kwiatkowska et al. (2006), a path of an MDP is a non-empty finite or infinite sequence of probabilistic transitions:

$$\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_0} \cdots$$

where $s_i \in S, a_i \in A(s_i)$ such that $P(s_i, a_i) > 0$ for all $i \in \mathbb{N}$. We denote $\pi(i)$ the $(i+1)th$ state of π and the final state of π as $last(\pi)$, if π is finite. We also define $IPaths_M$ and $FPaths_M$ and infinite and finite paths of M starting in the initial state $\bar{s}$.

MDPs are also annotated with rewards, which can be used to model a variety of quantitative measures of interest, in particular we have a reward of R_S which is accumulated when passing through state s and a reward of R_A when taking action a from state s.

In contrast to a path, an adversary or a strategy (also known as a scheduler or policy) represents a particular resolution of non determinism only. More precisely, an adversary is a function mapping every finite path π of M to a distribution $\sigma \in A(last(\pi))$, is a way of resolving the choice of action in each state, based on the MDP's execution so far.

Definition 3 *(Norman et al. 2017) An adversary* σ *of an MDP* $M = (S, \bar{s}, A, P, R)$ *is a function* $\sigma : FPaths_M \to Dist(A)$ *such that, for any* $\pi \in FPaths_M$*, we have* $\sigma(\pi)(a) > 0$ *only if* $a \in A(last(\pi))$*. Furthermore, let* Adv *denote the set of all adversaries.*

A memoryless strategy is one where its choices are dependant only on the current state and has a finite memory if it suffices to switch between a finite set of models. A strategy is also called deterministic if it always selects an action with probability 1. The behaviour under a given adversary σ is purely probabilistic and we can define a probabilistic measure $Prob^{\sigma}(M)$ over the set of paths $Path^{\sigma}(M)$. We now extend the Expected Reachability of DTMC to form the basis of our MDP rewards.

Expected Reachability For an adversary $\sigma \in Adv$, let

$$E^{\sigma}_{M}(F) \stackrel{def}{=} \int_{\pi \in Path^{\sigma}(M)} r(F, \pi)\, dProb^{\sigma}_{M}$$

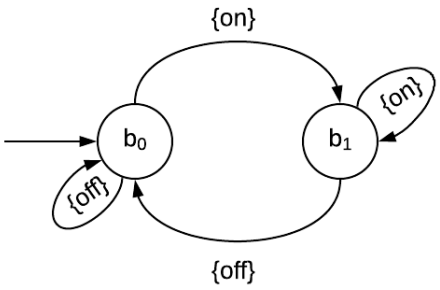
Definition 4 *For an MDP* $M = (S, \bar{s}, A, P, R)$*, the optimal maximum expected rewards of reaching a set of target states* $F \subseteq S$ *from the state* $s \in S$ *are defined as follows:*

$$E_M^{\max}(F)(s) \stackrel{def}{=} \sup_{\sigma \in Adv} E_M^{\sigma}(F)(s).$$

According to Kwiatkowska et al. (2006) computing values for expected reachability reduces to the stochastic shortest path problem for MDPs. A key result in this respect is that optimality with respect to expected reachability can always be achieved with deterministic and memoryless adversaries. This allows the use of efficient computational techniques such as policy iteration, which builds a sequence of strategies until an optimal one is reached, and value iteration, which computes increasingly precise approximations to the optimal probability or expected value. In PRISM, we compute this by using:

- **Rmax=? [F k=kmax+1]** – maximum expected accumulated reward/cost gained in k iterations;

Figure 4.2: *A bandit state transition with actions*



Example 2. From Figure 4.2, we extend our Example DTMC by turning probabilistic choices into non-deterministic by after one step, process has a non-deterministic choice between: (on) changing its state from b_0 to b_1 or remaining at b_1 if its already at b_1. Another non-deterministic choice between: (off) changing its state from b_1 to b_0 or remaining at b_0 if its already at b_0.

This can be modeled in PRISM as shown in Model 3.

In Model 4, we extend our RMAB problem from DTMC to an MDP to allow the scheduler make non-deterministic choices to the bandit it selects in each round. The behaviour of an MDP is as follows. Starting in the initial state $s = 0$, the scheduler selects an action non-determinstically to choose either bandit 1 or 2 based on the current states of the bandits and a reward of 1 is obtained if the chosen bandit is on. The turn is used the same way as the DTMC model to switch between changing the states of the bandits to choosing the bandits. Afterwards, the scheduler repeats the process from $s = 0$, and so forth.

Model 3 Example MDP

```
mdp //model type

module bandit1
   b1: [0..1];

   [off] b1=0 -> (b1'=0); //non-deterministic choice
   [on] b1=0 -> (b1'=1);

   [off] b1=1 -> (b1'=0);
   [on] b1=1 -> (b1'=1);
endmodule
```

Model 4 RMAB Problem in MDP

```
mdp

module bandit1
   ...
endmodule

module scheduler

     s : [0..2];
     turn : [-1..1];
     [initial] turn=-1 -> (turn'=0);

     [a1] turn=0 -> (turn'=1);
     [choose1] turn=1 -> (s'=1)&(turn'=0); //non-deterministic choice
     [choose2] turn=1 -> (s'=2)&(turn'=0);

endmodule

const int kmax; // bound on the steps

module counter
   ...
endmodule
...
```

4.3 Partially Observable Markov Decision Process

A limitation in our MDP model is that the scheduler can see the states of all the bandits which does not model the RMAB problem accurately, therefore we consider POMDPs which extend MDPs by restricting the extent to which only the current state of the bandit we pick can be observed, in particular by strategies that control them. In this project, we adopt the following notion of observability. Norman et al. (2017)

Definition 5 *A POMDP is a tuple* $M = (S, \bar{s}, A, P, R, O, obs)$ *where:*

- $(S, \bar{s}, A, P, R)$ *is an MDP;*
- O *is a finite set of observations;*
- $obs : S \to O$ *is a labelling of states with observations;*

such that, for any states s; $s' \in S$ *with* $obs(s) = obs(s')$, *their available actions must be identical, i.e.,* $A(s) = A(s')$.

The current states of a POMDP cannot be directly determined, only the corresponding observation $obs(s) \in O$. The requirement on available actions in the above definition follows from the fact that, if states have different actions available, then they are not observationally equivalent as the available actions are not hidden, and hence should not have the same observation. The notions of paths, strategies and probability measures given above for MDPs transfer directly to POMDPs. However, the set Adv of all strategies for a POMDP M only includes observation-based strategies defined below.

Definition 6 *Observation-based strategy A strategy of a POMDP* $M = (S, \bar{s}, A, P, R, O, obs)$ *is a function* $\sigma : FPaths_M \to Dist(A)$ *such that:*

- *is a strategy of the MDP* $(S, \bar{s}, A, P, R)$;
- *for any paths* $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots s_n$ *satisfying* $obs(s_i) = obs(s'_i)$ *and* $a_i = a'_i$ *for all* i, *we have* $\sigma(\pi) = \sigma(\pi')$, *where* $\pi' = s'_0 \xrightarrow{a'_0} \cdots s'_n$.

Let Adv *denote the set of all (observation-based) strategies of* M.

We now introduce rewards for POMDP by extending definition 4 for expected reachability to use observables denoted by $E^{\sigma}_{M}(F\ O)$.

Definition 7 *For a POMDP* $M = (S, \bar{s}, A, P, R, O, obs)$, *the optimal maximum expected rewards of reaching a target* O *(i.e., a set of states for an MDP and a set of observations for a POMDP) from the state* $s \in S$ *is defined as follows:*

$$E^{\max}_{M}(F\ O) \stackrel{def}{=} \sup_{\sigma \in Adv} E^{\sigma}_{M}(F\ O).$$

We know that in MDPs we can achieve optimal values (see Definition 3), but in the case of POMDPs, this no longer holds. In fact, determining the optimal probabilities and expected rewards defined above is undecidable, making exact solution intractable. Instead, the optimal value can be approximated, for example via analysis of the belief MDP and POMDP.

Beliefs. Given a POMDP M, a belief MDP $B(M)$ is an equivalent MDP, whose (continuous) state space comprises beliefs, which are probability distributions over the state space of M. Intuitively, although we may not know which of several observationally-equivalent states we are currently in, we can determine the likelihood of being in each one, based on the probabilistic behaviour of M. See Definition 5 given by Norman et al. (2017) for a more detailed explanation. The optimal values for the expected reward to reach a target in the belief MDP equal those for the POMDP, which is formally stated by the following proposition.

Proposition 1. If $M = (S, \bar{s}, A, P, R, O, obs)$ is a POMDP where $O \in O$ a set of observations, then:

$$E_M^{max}(F\ O) = E_{B(M)}^{max}(F\ T_O)$$

where $T_O = \{b \in Dist(S) \mid \forall_s \in S\}$

Using Model 5, we will now extend our MDP to limit the perfect knowledge of the system, which is achieved by transforming it to a POMDP such as there is a realistic model where our system can only observe the current state of the bandit it picks. This is done is PRISM through the keyword observables and where the unspecified variables are considered hidden. The observable values $v1$ and $v2$ are used to observe the current state of bandit 1 and 2 respectively when it is played. We also extend our scheduler to observe only the current value of the bandit it picks using these variables, hiding states of the other unpicked bandits, which is used in our belief probabilities. Our system now can only see the current states of the observables variables we mention that is the current state of our scheduler s, the $turn$ our scheduler is on and k which is the iteration we are on. In addition to that, by assigning the observable variable $v1$ and $v2$ to the state of each bandit, the system only know the current state of the bandit it picks and cannot see the rest.

As we are using an extension to PRISM for our POMDP that uses bounds, to analyse we use an MDP-based abstraction. This provides a lower bound on the minimum probability or an upper bound on the maximum which forms outer bounds on reachability probabilities. From Jamieson and Nowak (2014), we know that despite the undecidability of the POMDP problems, useful results can be obtained, often with precise bounds. We also refine our expected cumulative reachability property by increasing grid resolution. Although, just incrementing the resolution is not guaranteed to yield tighter bounds and in fact can yield worse bounds.

Our model operates by maintaining a belief state, $b(s)$, which is a discrete probability distribution over the states in which the environment might exist where b_0 is the initial belief state. After each action is taken and a new observation is made, the belief state is updated. To choose an action, the model refers to an adversary which is a mapping from belief state to action. The optimal adversary is one which will maximise the rewards obtained.

Model 5 RMAB Problem in POMDP

```
pomdp

observables
//v1 and v2 are when the scheduler picks bandit1 or 2 it can observe
    them.
     s, turn, k, v1, v2
endobservables

module bandit1
...

module scheduler

     s : [0..2];
     turn : [-1..1];
     v1 : [0..1];
     v2 : [0..1];

     [initial] turn=-1 -> (turn'=0);
     [a1] turn=0 -> (turn'=1);
     //v1 and gets the value of bandit 1 and 2 respectively
     //so the scheduler can look at the current state
     //of the bandit it picks
     [choose1] turn=1 -> (s'=1)&(turn'=0)&(v1'=b1);
     [choose2] turn=1 -> (s'=2)&(turn'=0)&(v2'=b2);

endmodule

...
```

4.4 Chapter Summary

In this chapter, we looked at the three models and how they differentiate from each other where a DTMC has no control over the selection and exhibits probabilistic choices whereas an MDP exhibits both non deterministic choices and probabilistic choices but the limitation is that an MDP assumes perfect knowledge of the system which does not model our RMAB problem, therefore, an extension to MDP, another model called POMDP was introduced to hide the states of the non-played arms. We also discussed PRISM more broadly and how each of the models were developed in the tool and the property we use to analyse.

The next chapter will give the analysis of the optimal strategies formed by the POMDP model.

5 Analysis

This chapter first discusses the additional tools developed which are used to evaluate the system. The main purpose of this chapter is to analyse the three models (DTMC, MDP and POMDP) developed in Section 3 using the tools.

5.1 Tools Developed

There were two types of tools developed for the purpose of this project:

1. **Model Creation**
   (a) To automate the model building part of the project, several Python scripts were developed which can be used to create the PRISM model files for our POMDP model in different ways, including manual input of the probability distribution of the bandits or to create models from reading a file which can have different number of bandits and probability distributions.
   (b) As the purpose of this project was also extended to comparing the cumulative reward acquired by each model (DTMC, MDP and POMDP), another script for the sole purpose of creating the three models with the given probability distribution of the bandits was developed using Python.

2. **Graph Generation**
   (a) As the extension of PRISM we use to analyse our POMDP model does not currently support the generation of the adversary graphs, but can output the states, transitions and belief probabilities into a simple text file. A snippet of the file is provided in Figure 5.1 which is not understandable to a normal reader. The output generated is in the format:

   current state number:(values of visible variables),[belief probabilities for values of hidden variables] & non-deterministic number/probability @ (new values of visible variables) > state number they transition to:(new values of visible variables),[new belief probabilities for values of hidden variables]

   To make this more readable, we transformed the file into a graph which was developed by using a Python script to read in the file and along with pygraphviz (Hagberg et al. 2004) we created our strategy graphs which will be used to analyse optimal strategies.
   (b) Another Python script with the help of matplotlib (Hunter 2007) was developed to create the comparison graph provided the results of the three models which PRISM can output into a text file. The reward comparison graph consists the rewards obtained from 0 through some *kmax* iterations by all the three models (DTMC, MDP and POMDP) to compare the results.

```
0:(0,-1,0,0,0),[1.0, 0.0, 0.0, 0.0] & 0/1.0 @ (0,0,0,0,0) > 1:(0,0,0,0,0),[0.25, 0.25, 0.25, 0.25]
1:(0,0,0,0,0),[0.25, 0.25, 0.25, 0.25] & 0/1.0 @ (0,1,0,0,0) > 2:(0,1,0,0,0),[0.25, 0.25, 0.25, 0.25]
2:(0,1,0,0,0),[0.25, 0.25, 0.25, 0.25] & 1/0.5 @ (2,0,0,1,1) > 3:(2,0,0,1,1),[0.0, 0.5, 0.0, 0.5]
2:(0,1,0,0,0),[0.25, 0.25, 0.25, 0.25] & 1/0.5 @ (2,0,0,0,1) > 4:(2,0,0,0,1),[0.5, 0.0, 0.5, 0.0]
3:(2,0,0,1,1),[0.0, 0.5, 0.0, 0.5] & 0/1.0 @ (2,1,0,1,1) > 5:(2,1,0,1,1),[0.3, 0.2, 0.3, 0.2]
4:(2,0,0,0,1),[0.5, 0.0, 0.5, 0.0] & 0/1.0 @ (2,1,0,0,1) > 6:(2,1,0,0,1),[0.2, 0.3, 0.2, 0.3]
5:(2,1,0,1,1),[0.3, 0.2, 0.3, 0.2] & 0/0.5 @ (1,0,1,1,2) > 7:(1,0,1,1,2),[0.0, 0.0, 0.6, 0.4]
5:(2,1,0,1,1),[0.3, 0.2, 0.3, 0.2] & 0/0.5 @ (1,0,0,1,2) > 8:(1,0,0,1,2),[0.6, 0.4, 0.0, 0.0]
6:(2,1,0,0,1),[0.2, 0.3, 0.2, 0.3] & 1/0.6 @ (2,0,0,1,2) > 9:(2,0,0,1,2),[0.0, 0.5, 0.0, 0.5]
6:(2,1,0,0,1),[0.2, 0.3, 0.2, 0.3] & 1/0.4 @ (2,0,0,0,2) > 10:(2,0,0,0,2),[0.5, 0.0, 0.5, 0.0]
7:(1,0,1,1,2),[0.0, 0.0, 0.6, 0.4] & 0/1.0 @ (1,1,1,1,2) > 11:(1,1,1,1,2),[0.24, 0.26, 0.24, 0.26]
8:(1,0,0,1,2),[0.6, 0.4, 0.0, 0.0] & 0/1.0 @ (1,1,0,1,2) > 12:(1,1,0,1,2),[0.24, 0.26, 0.24, 0.26]
9:(2,0,0,1,2),[0.0, 0.5, 0.0, 0.5] & 0/1.0 @ (2,1,0,1,2) > 13:(2,1,0,1,2),[0.3, 0.2, 0.3, 0.2]
10:(2,0,0,0,2),[0.5, 0.0, 0.5, 0.0] & 0/1.0 @ (2,1,0,0,2) > 14:(2,1,0,0,2),[0.2, 0.3, 0.2, 0.3]
```

Figure 5.1: *PRISM generated adversary file*

5.2 The Strategies

We now run our model against our specified property in PRISM to get bound results and the strategies. The PRISM results we look at are given in Figures 5.2 and 5.3, which first shows the variables in the models, where the variables in the round brackets are hidden and the rest are observables. The second figure shows the Grid Statistics which are used in our bounds, where we want to have the bounds as similar as possible so the optimal value can be approximated. It also shows the result of the property i.e the cummulated reward obtained which is used in our reward comparison graph.

```
Type:        POMDP
Modules:     bandit1 bandit2 scheduler counter
Variables:   (b1) (b2) s turn v1 v2 k
```

Figure 5.2: *PRISM snippet of the model being used*

```
Grid statistics: resolution=20, points=25383, unknown points=25215

Outer bound: 2.7
Inner bound: 2.7

Value in the initial state: 2.7

Time for model checking: 1.618 seconds.

Result: 2.7 (value in the initial state)
```

Figure 5.3: *PRISM snippet of the results we evaluate*

Using Figure 5.4 and 5.5, we will try to explain a adversary graph which is easy to read for POMDP and the result comparison graph for each model first so we can try to analyse optimal strategies for other probability distributions which generates are a bit more complex. We also define the chance of success of each bandit as them being at value 1 or are turned “on". The probability distribution of the bandits are as follows:

$$P_1 = \begin{matrix} 0{:}5 & 0{:}5 \\ 0{:}5 & 0{:}5 \end{matrix} \quad \& \quad P_2 = \begin{matrix} 0{:}1 & 0{:}9 \\ 0{:}1 & 0{:}9 \end{matrix}$$

A state represents a change in the scheduler state at every time step where the value of s represents the current state of the scheduler. The variable $turn$ will change its value from 0 to 1 and back representing turning states of the bandit and picking a bandit respectively except for state number 0 as in that state we initialise our bandits with a random chance of being “on" or “off". The variables $v1; v2:::$ will represent the current state of the bandit picked and the system will run for k iterations. To explain the components to the belief probabilities, we take an example belief probability from state number 2, which are [0.05, 0.45, 0.05, 0.45]. The Table 5.1 shows that when the values of both bandits is 0 then the belief probability is 0.05. As we can see from the

The bandits	$b1$ & $b2$	$b1$ & $b2$	$b1$ & $b2$	$b1$ & $b2$
States of the bandits	0 & 0	0 & 1	1 & 0	1 & 1
Belief probabilities	0.05	0.45	0.05	0.45

Table 5.1: *Table to show how belief probabilities relate to bandits*

graph (Figure 5.5), the optimal strategy is to always pick bandit 2 (the one with 90% chance of being at value 1) every turn to maximise the reward. The graph here only shows for $k = 3$, but the strategy is optimal for any k. This is because at every state we have a 90% chance of bandit 2 having the value 1, and there is no need to explore. We can also see the belief probability at $turn$ 1 is 0.45 for both components where bandit 2 is at value 1 compared to 0.05 chance for bandit 1 having value 1, even when bandit 2 does not gain a reward. Now considering the optimal expected reachability, we obtain the following results as the grid resolution (GR) for $kmax = 10$ which is refined during the analysis:

- GR = 2 yields 953 grid points and the bounds [9.0, 9.499]
- GR = 3 yields 1737 grid points and the bounds [9.0, 8.999]
- GR = 4 yields 2871 grid points and the bounds [9.0, 8.999]

Finally, we obtained a precise bound of ([9.0, 9.0]) with grid resolution = 5 for which the number of points in the grid were 4425 with 4377 unknown points.

We now compare the rewards obtained from POMDP with the other models to compare and evaluate the optimal strategy using Figure 5.4 shown below. The reward comparison graph clearly shows that the random probabilistic choice for the DTMC, we obtain less than what we can gather from POMDP which is very close to what we could have gathered from perfect knowledge of states (MDP). This is expected as the MDP always chooses the bandit which is "on" and since bandit 2 has such a high success chance, even the random choices of DTMC can yield a high reward but our POMDP performs better than that by learning about and exploiting the system.

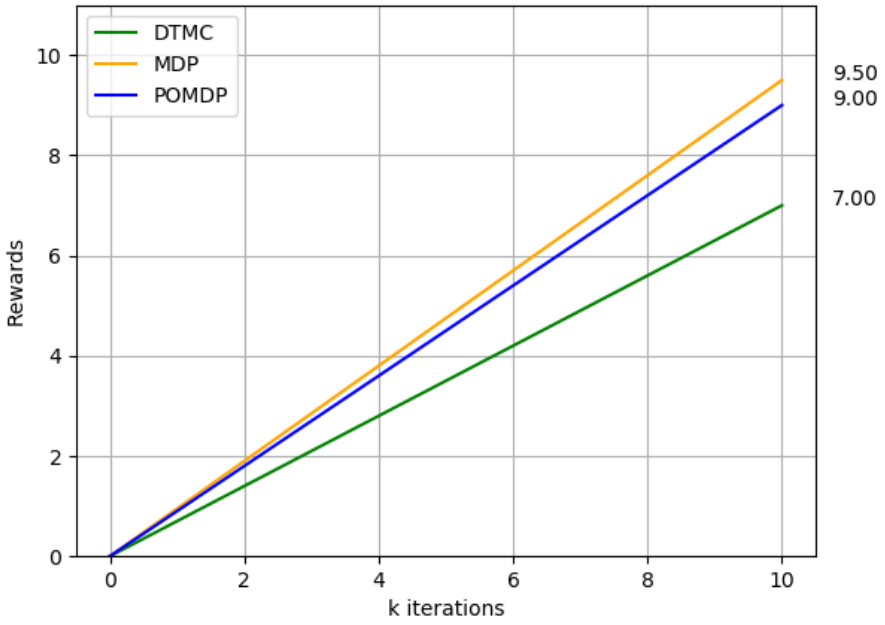


Figure 5.4: *Max Rewards of each model (2 bandits | 90% success) with respect to k iterations*

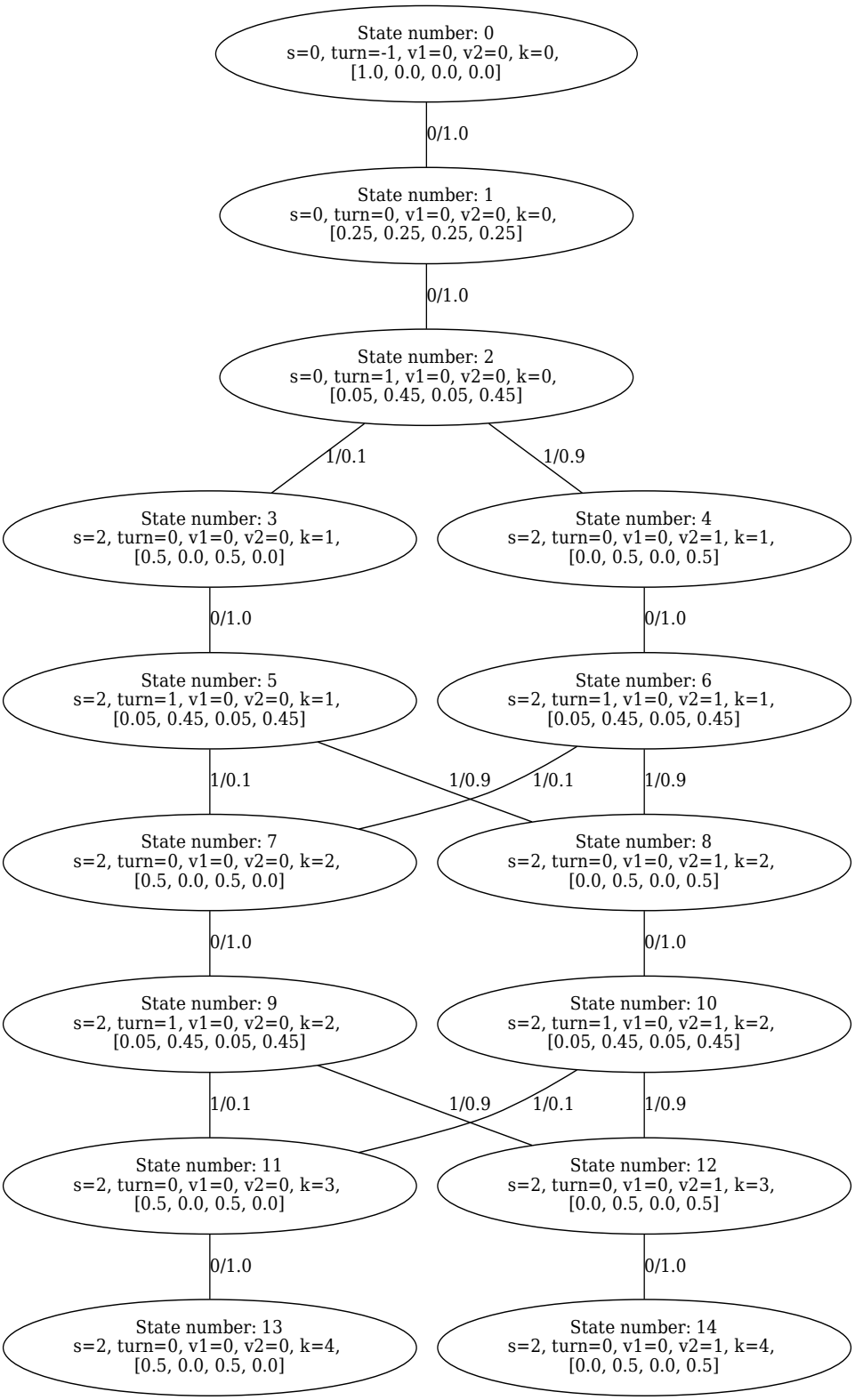


Figure 5.5: *POMDP of 2 bandit with one having 90% success rate*

5.2.1 Two-armed Bandits

We now start to look at probability distributions which do not have that high of a success chance so we have to consider exploring the system, by first analysing two bandits which have very similar chance of success using Figure 5.3 given below.

The probability matrix distribution for bandits in Figure 5.3 is as follows:

$$P_1 = \begin{matrix} 0{:}5 & 0{:}5 \\ 0{:}5 & 0{:}5 \end{matrix} \quad \& \quad P_2 = \begin{matrix} 0{:}4 & 0{:}6 \\ 0{:}6 & 0{:}4 \end{matrix}$$

Figure 5.8 shows the optimal strategies for $kmax = 3$, and our analysis for $kmax = 5$ follows. The optimal strategy given is to first pick bandit 2 as it has a higher 60% chance of success. If a reward is given, then we should switch to bandit 1 which now has a 50% probability of success as oppose to 40%. Now, the optimal strategy is to switch back to bandit 2 regardless of bandit 1's reward as there is a 52% chance of success with a very slight difference of 2% between having the value of 1 for bandit 1 and bandit 2 (higher). If there is no reward from bandit 2 in the first place then we should still keep picking bandit 2. This tells us that it is always better to explore for bandit 1 and also when a reward is received by bandit 2. It is also better to exploit the bandit 2 when no reward is obtained as it still has a higher probability of success.

we obtain the following results as the grid resolution (GR) for $kmax = 10$ which is refined during the analysis:

- GR = 3 yields 1737 grid points and the bounds [5.331, 5.833]
- GR = 5 yields 4425 grid points and the bounds [5.331, 5.499]
- GR = 15 yields 58465 grid points and the bounds [5.331, 5.354]

Finally, we obtained a precise bound of ([5.331, 5.331]) with grid resolution = 20 for which the number of points in the grid were 125735 with 125567 unknown points.

These states keep repeating, and therefore we now create a simple figure based on the different states achieved shown in Figure 5.6. We also present our reward comparison graph of the three models which is shown in Figure 5.7. This is as expected for the same reasons as before although our POMDP is closer to the DTMC and this might be because of the exploration-exploitation trade-off where some rewards were potentially sacrificed in order to learn.

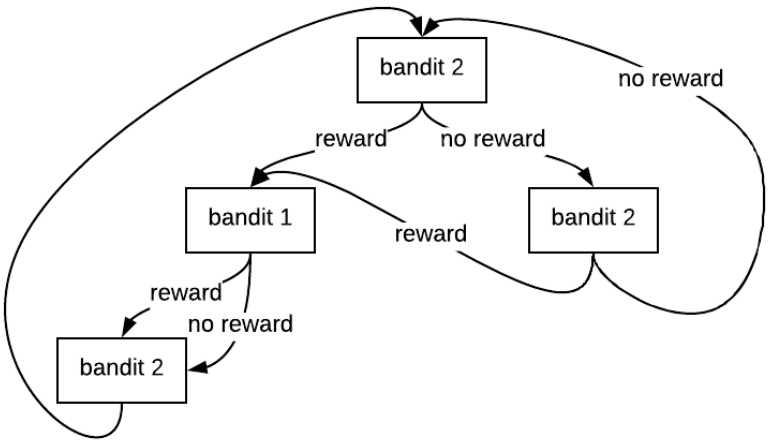


Figure 5.6: *Representation of the optimal strategy*

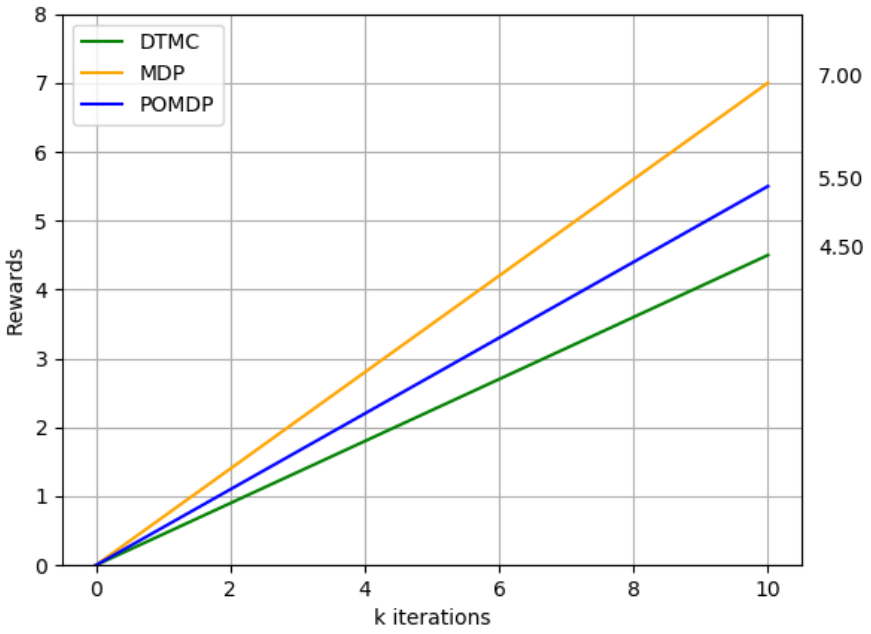


Figure 5.7: *Max Rewards of each model (2 bandits | 60% success) with respect to k iterations*

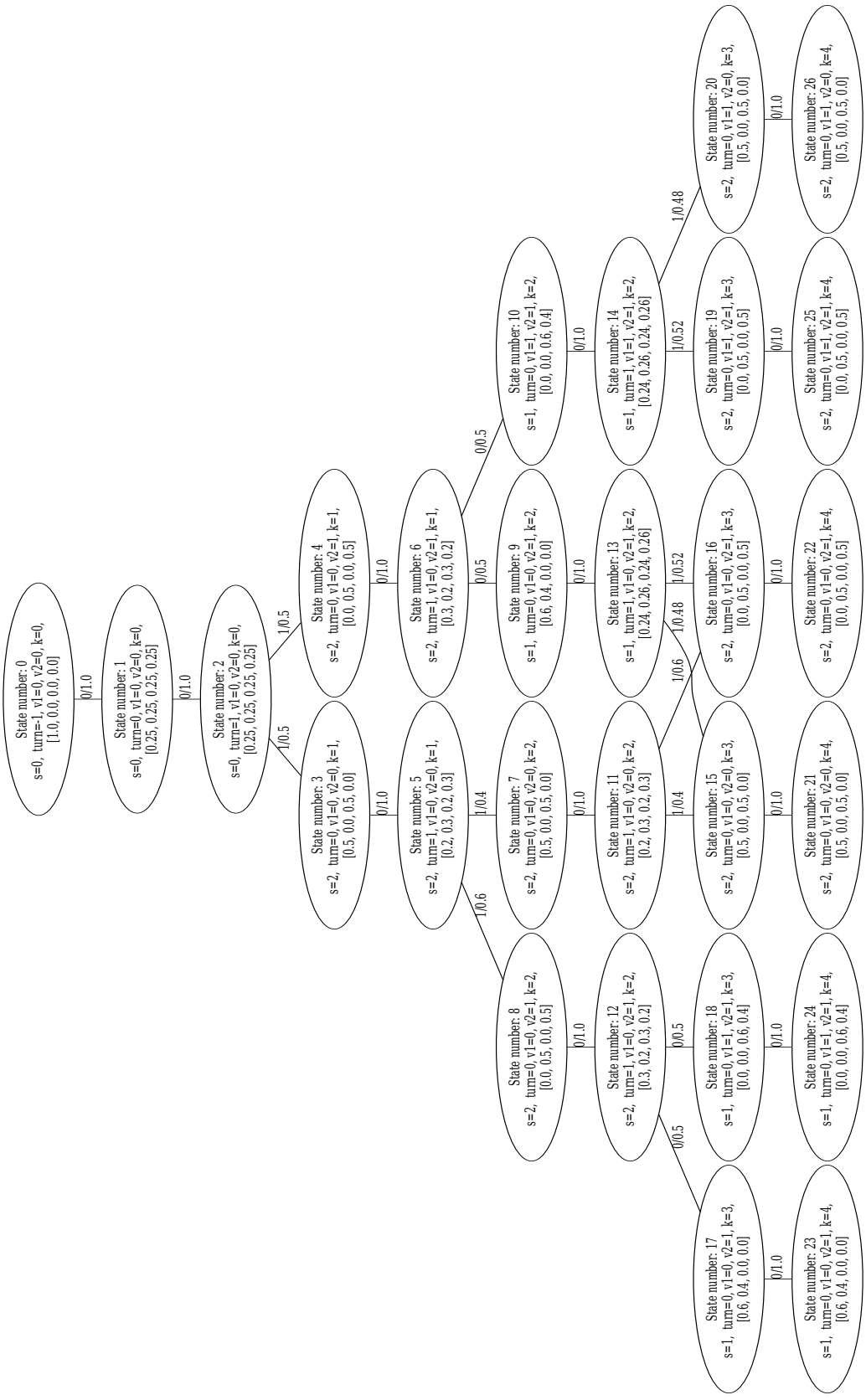


Figure 5.8: *POMDP of 2 bandits with one having 60% success rate (graph is rotated to fit)*

We now look at another two-armed bandit with different probability distributions as follows:

$$P_1 = \begin{matrix} 0{:}6 & 0{:}4 \\ 0{:}4 & 0{:}6 \end{matrix} \quad \& \quad P_2 = \begin{matrix} 0{:}3 & 0{:}7 \\ 0{:}6 & 0{:}4 \end{matrix}$$

Figure ?? shows the optimal strategy for the above probability distributions. The strategy given is to choose bandit 2 which has a 55% chance of success and then choose bandit 1 if a reward was obtained as there is a higher chance than the 30% success rate bandit 2 provides. Now, if bandit 1 gives a reward then we should keep on choosing it as there is a 60% chance of success until there is no reward obtained, from which we should choose bandit 2 with a 52.6% chance of success. The reasoning behind is to keep on exploring so that scheduler can learn and improve the success for future. Now, if there was no reward obtained by bandit 2 in the first place then we should keep on choosing as there is still a 70% chance of success until a reward is obtained, from which we should switch to bandit 1. The reasoning is to exploit the higher bandit as it has a higher chance and then explore as the success chance drops.

we obtain the following results as the grid resolution (GR) for $kmax = 10$ which is refined during the analysis:

- GR = 3 yields 1737 grid points and the bounds [5.749, 6.181]
- GR = 5 yields 4425 grid points and the bounds [5.749, 5.892]
- GR = 15 yields 58465 grid points and the bounds [5.755, 5.789]

Finally, we obtained a precise bound of ([5.7555, 5.758]) with grid resolution = 35 for which the number of points in the grid were 593545 with 593257 unknown points.

In figure 5.9 we show the strategy in a simple format and also present our reward comparison graph of the three models which is shown in Figure 5.10.

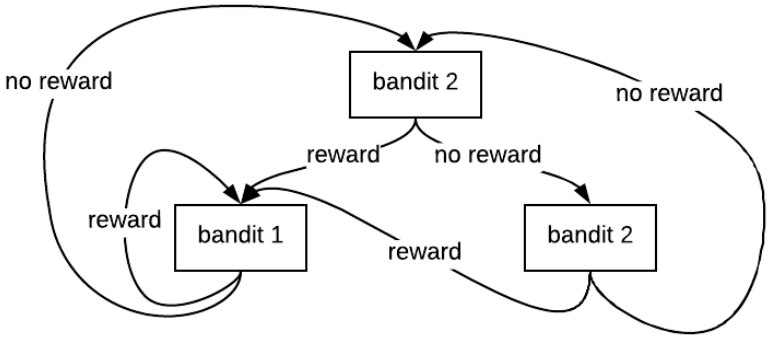


Figure 5.9: *Representation of the optimal strategy*

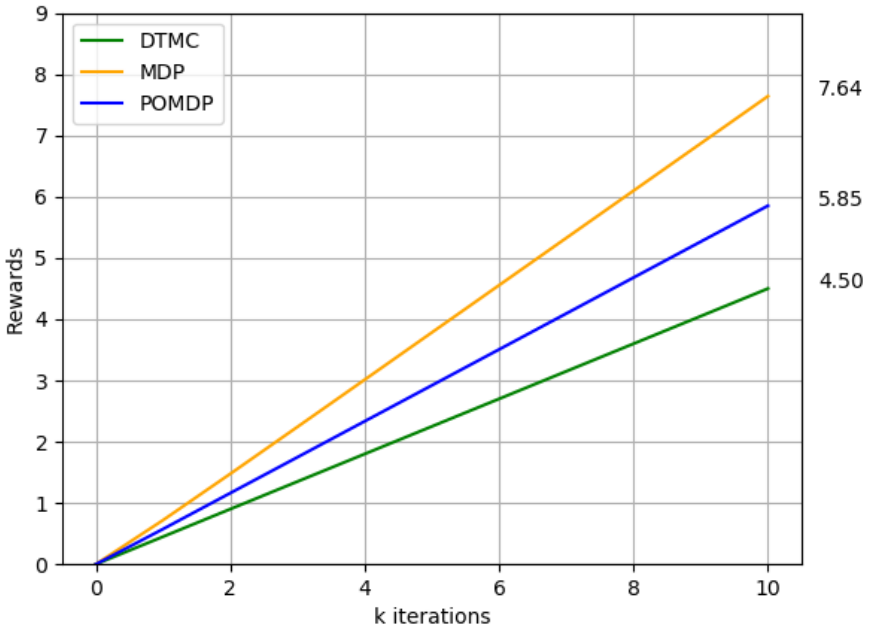
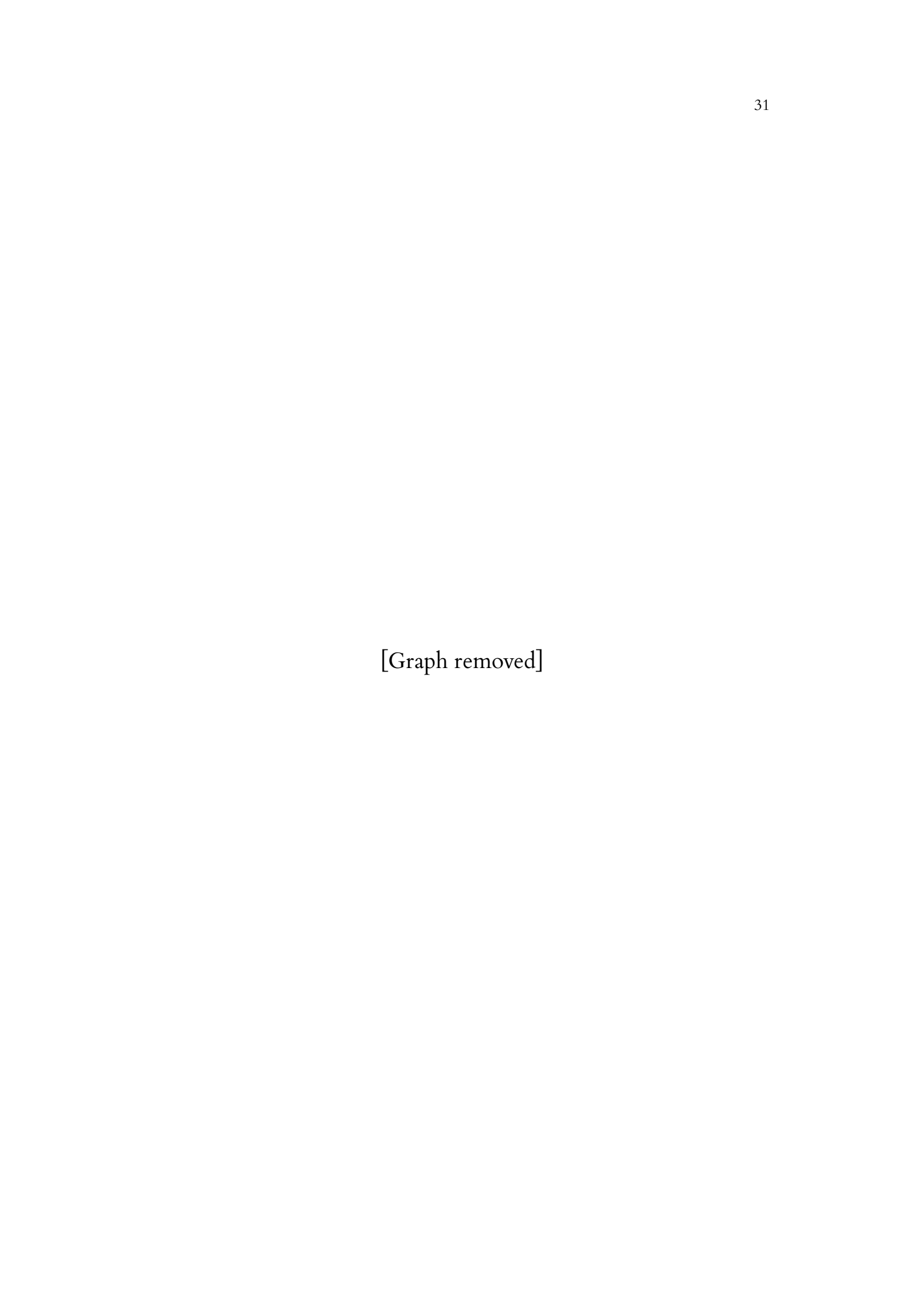


Figure 5.10: *Max Rewards of each model (2 bandits) with respect to k iterations*


[Graph removed]

5.2.2 Three-armed Bandits

We now extend our RMAB problem by having three bandits and analyse their optimal strategies. The probability distribution for the three bandits we look at is:

$$P_1 = \begin{matrix} 0{:}5 & 0{:}5 \\ 0{:}5 & 0{:}5 \end{matrix} \quad \& \quad P_2 = \begin{matrix} 0{:}6 & 0{:}4 \\ 0{:}4 & 0{:}6 \end{matrix} \quad \& \quad P_3 = \begin{matrix} 0{:}3 & 0{:}7 \\ 0{:}6 & 0{:}4 \end{matrix}$$

The strategy given is to choose bandit 3 with a 55% chance of success, after which if a reward is given then switch to bandit 2 with 50% chance of success. Now, if bandit 2 gives an immediate reward then we should switch back to bandit 3 with a 58% success chance repeating the whole process. The reason behind is that we are exploiting the higher success chance bandits because of immediate rewards. However, if no reward is obtained at that time then we should pick bandit 3 and keep choosing it if there is no reward, although now, if there is a reward obtained we should pick bandit 1 which has a 50% chance of success. This exploration is to allow the scheduler learn for future chances of success. Due to the complexity of the graph in both time and understand-ability, the strategy after bandit 1 is not observed. On the other hand, if bandit 3 does not gain a reward in the first place then we should still keep on choosing until a reward is obtained and switch to bandit 2. The reasoning is to exploit the best bandit we have until a reward is obtained from it.

we obtain the following results as the grid resolution (GR) for $kmax = 5$ which is refined during the analysis:

- GR = 3 yields 11281 grid points and the bounds [2.873, 3.266]
- GR = 5 yields 65329 grid points and the bounds [2.872, 3.042]
- GR = 8 yields 495991 grid points and the bounds [2.871, 2.918]

Finally, we obtained a close bound (as GR > 13 is becoming computationally heavy) of ([2.874, 2.911]) with grid resolution = 13 for which the number of points in the grid were 5803681 with 5790241 unknown points.

In figure 5.11 we show the strategy in a simple format and also present our reward comparison graph of the three models which is shown in Figure 5.12.

Note: The adversary graphs are not shown anymore due to not fitting them in a page and are provided as part of submission.

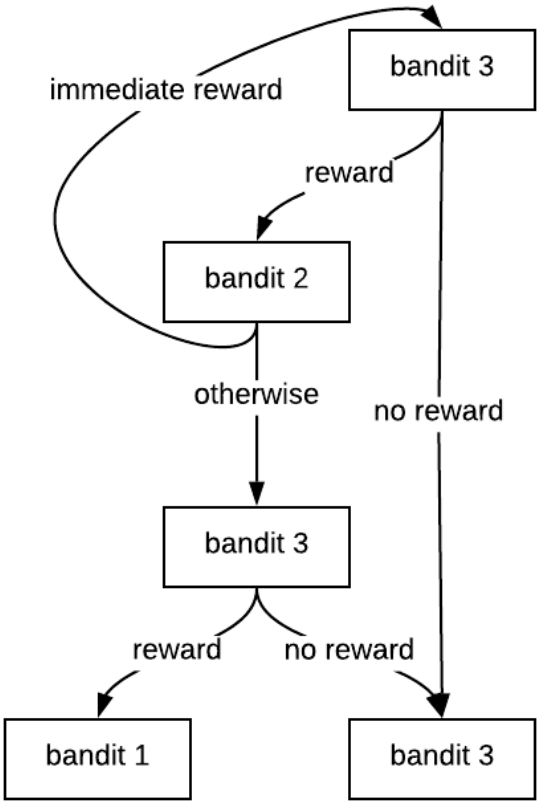


Figure 5.11: *Representation of the optimal strategy*

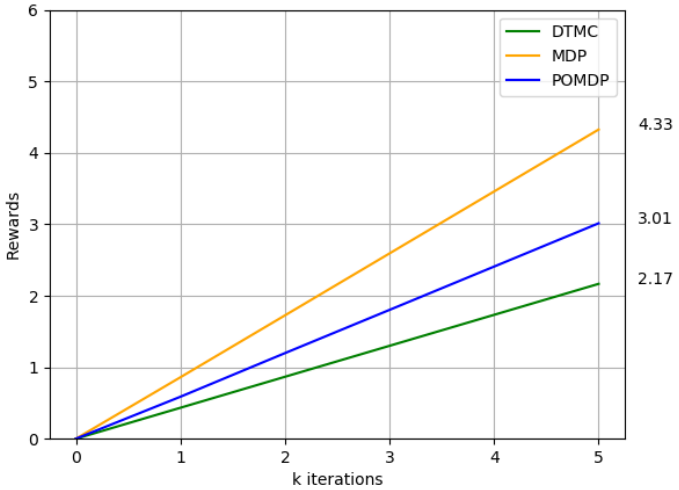


Figure 5.12: *Max Rewards of each model (3 bandits) with respect to k iterations*

We now look at another three-armed bandit with different probability distributions as follows:

$$P_1 = \begin{matrix} 0{:}55 & 0{:}45 \\ 0{:}45 & 0{:}55 \end{matrix} \quad \& \quad P_2 = \begin{matrix} 0{:}6 & 0{:}4 \\ 0{:}4 & 0{:}6 \end{matrix} \quad \& \quad P_3 = \begin{matrix} 0{:}3 & 0{:}7 \\ 0{:}6 & 0{:}4 \end{matrix}$$

The strategy given is to choose bandit 3 with a 55% chance of success and if there is a reward obtained then we should choose bandit 2 with a 50% chance of success. If bandit 2 now gains a reward then bandit 3 should be picked. After that, we should stay on bandit 3 until a reward has been gained, after which we should switch to bandit 2 again and repeat the process. The reason is to exploit the higher chance of success bandits to get rewards. However, if there was no reward at bandit 2 then we switch to bandit 3 again and remain there until a reward has been gained but should switch to bandit 1 with a 50% chance of success. Next, we should switch back to bandit 3 regardless of the reward from bandit 1. This exploration is done to have the scheduler learn for future chances of success. Now, if there was no reward obtained from bandit 3 at the start then we should keep choosing bandit 3 until a reward is obtained from where this whole process is repeated. The reason behind this exploitation is that bandit 3 is our highest chance of success bandit and should be exploited until a reward is gained.

- GR = 3 yields 11281 grid points and the bounds [2.873, 3.266]
- GR = 5 yields 65329 grid points and the bounds [2.870, 3.056]
- GR = 8 yields 495991 grid points and the bounds [2.873, 2.918]

Finally, we obtained a close bound (as GR > 13 is becoming computationally heavy) of ([2.871, 2.916]) with grid resolution = 13 for which the number of points in the grid were 5803681 with 5790241 unknown points.

In figure 5.13 we show the strategy in a simple format and also present our reward comparison graph of the three models which is shown in Figure 5.14.

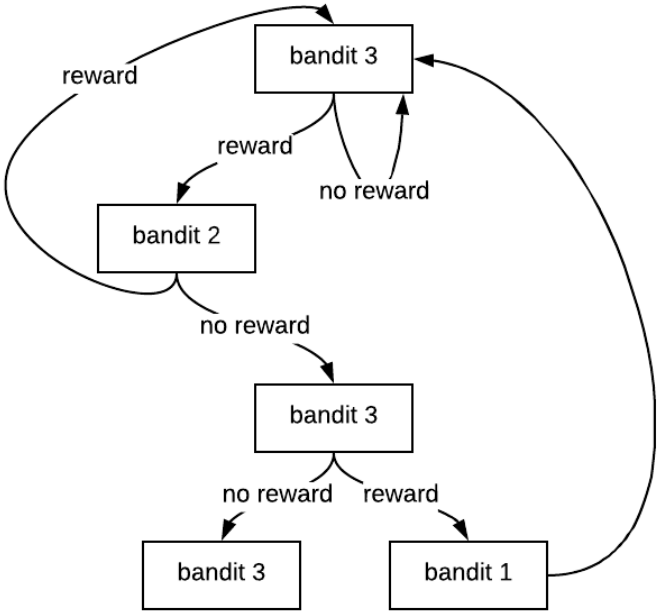


Figure 5.13: *Representation of the optimal strategy*

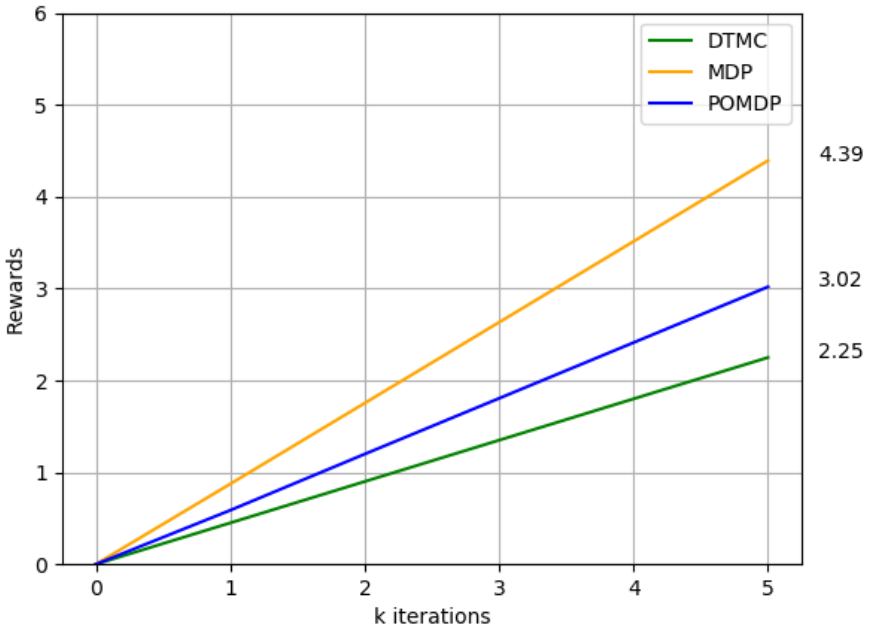


Figure 5.14: *Max Rewards of each model (3 bandits) with respect to k iterations*

5.2.3 Four-armed Bandits

We now extend our RMAB problem by having four bandits and analyse their optimal strategies. The probability distribution for the four bandits we look at is:

$$P_1 = \begin{matrix} 0{:}5 & 0{:}5 \\ 0{:}5 & 0{:}5 \end{matrix} \quad \& \quad P_2 = \begin{matrix} 0{:}6 & 0{:}4 \\ 0{:}4 & 0{:}6 \end{matrix} \quad \& \quad P_3 = \begin{matrix} 0{:}3 & 0{:}7 \\ 0{:}6 & 0{:}4 \end{matrix} \quad \& \quad P_4 = \begin{matrix} 0{:}55 & 0{:}45 \\ 0{:}45 & 0{:}55 \end{matrix}$$

The problem faced in analysing the optimal strategies is due to high computational cost, the model was not able to build due to time and memory constraints and we could only gather results for $kmax = 2$ and the comparison is shown in Figure 5.15 and a snippet of the PRISM output is shown in Figure 5.16.

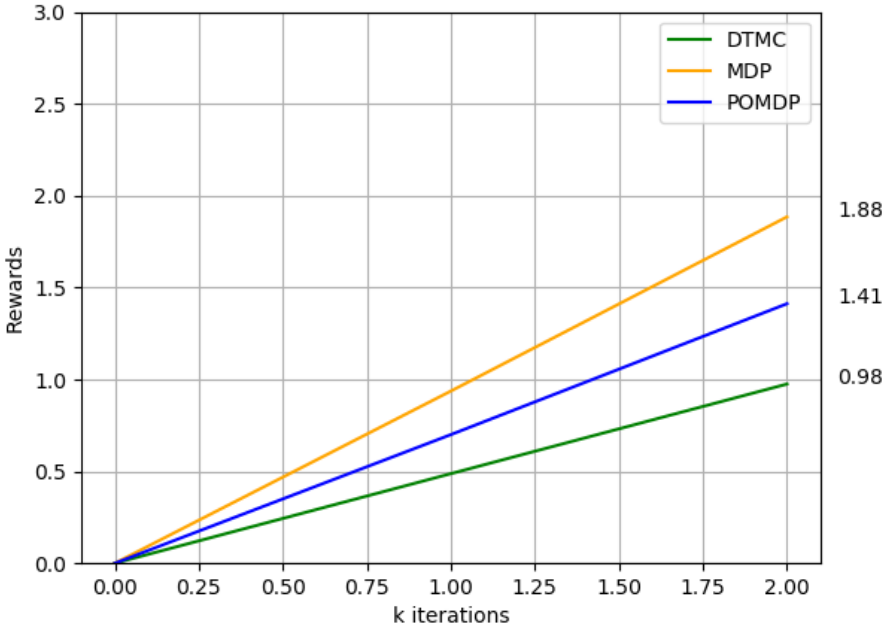


Figure 5.15: *Max Rewards of each model (4 bandits) with respect to k iterations*

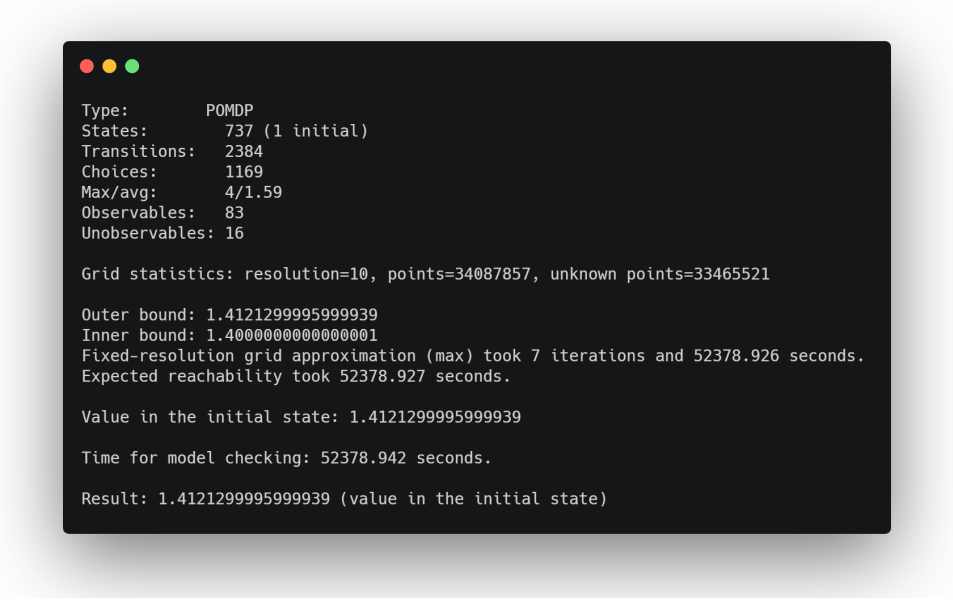


Figure 5.16: *Max Rewards of each model (4 bandits) with respect to k iterations*

5.3 Chapter Summary

In this chapter, we shed some light on the tools that were developed to automate model creation and also to develop the strategy and reward comparison graphs to analyse the models. We also then analysed the behaviour of some probability distributions with two-armed bandits and three-armed bandits and also discussed why were those the optimal strategies and the expectation from the reward comparison graphs. We also looked at the four-armed bandit which could not be analysed due to them being so computationally heavy. The next chapter will conclude the project and give some ideas for future work.

6 Conclusion

6.1 Summary of the project

This dissertation's primary goal was to model and analyse the optimal strategies for the Restless Multi-armed Bandit problem. We first looked at the problem itself and its application in the real world and how probabilistic model checking tools such as PRISM can help model and analyse the RMAB problem. We then defined the three probabilistic models: DTMC, MDP and POMDP; and how they can be constructed in PRISM. We also defined temporal logics that formed our reward measure and the technique used to analyse was belief space approximation.

We then discussed the tools developed for this project to automate model creation in PRISM and also construct strategy graphs from the results given by it. Experiments involving other PRISM models for POMDP were used as the inspiration for the analysis and the experiments. Several probability distributions with different number of armed-bandits were then developed as PRISM models with the help of the automated tools that were implemented. The PRISM results were then evaluated and developed into adversary graphs for the analysis of POMDP strategies and reward comparison graphs which helped to compare the three models. We also then analysed those optimal strategies and gave the reasoning behind them and showed how the exploration-exploitation trade-off problem affected rewards. I believe that model checking tools such as PRISM are useful to model and analyse real world problems such as the RMAB problem.

6.2 Future Work

Since this work deals with a development version of PRISM, there leaves plenty of opportunity for running these experiments again in the future with other more efficient approximation techniques and zone based implementations. We can also then extend our experiments to model real world probability distributions and would be int resting to see how the given optimal strategies by PRISM differentiates from the real world experiments we can run.

A Obtaining and Running PRISM Extension

The PRISM extension used for this project can be found on: https://github.com/prismmodelchecker/prism-ext/tree/pomdps

This can be built in exactly the same way as normal versions of PRISM. See the installation on: http://www.prismmodelchecker.org/manual/InstallingPRISM/Instructions.

This can be done using the following:

```
git clone https://github.com/prismmodelchecker/prism-ext.git
cd prism-ext/prism
git checkout pomdps
make
```

Now the format for running PRISM is: prism #model file# #property file# -prop #property number#

We also add some other switches, namely:

-const, -gridresolution, -javamaxmem, -exportadv and -exportresults

An example:

```
prism 2bandit50.prism bandits.prop -prop 1 -const kmax=10
    -gridresolution 10 -exportadv 2bandit50.adv -exportresults
    2bandit50.res
```

This will generate two files (2bandit50.adv and 2bandit50.res) which are the strategy with states and the other is the reward achieved for kmax = 10. These will be used in the tools mentioned below to generate the adversary graph and the reward comparison graph.

Note: -javamaxmem switch is used for bandits which require more computational power and higher values of kmax.

B Project Directory Structure

```
root
├── Adversary Graphs
│   └── Contains the adversary graphs analysed in this project
├── Basic Models
│   └── Contains the basic models for dtmc, mdp and pomdp as references
├── Models analysed
│   └── Contains all the models developed for this project
├── Reward Graphs
│   └── Contains all the reward comparison graphs for this project
└── Tools
    └── Contains all the tools developed for this project
```

The model names correspond to the order they appear in:

1. 2bandit90
2. 2bandit50
3. 2banditRandom
4. 3bandit50
5. 3banditRandom
6. 4bandit

C Using the Tools

As mentioned there are four tools developed.

1. Make the prism model files given the probability distribution. This can be achieved in two ways, either use generateModelsFromInput.py to manually input the model and the probability distributions or use generateModelsFromText.py with a text file like the example modelText provided.
   ```
   > python generateModelsFromInput.py
   > python generateModelsFromText.py modelText.txt
   ```
2. Make the three models given one probability distribution. This is achieved using the file generateDuplicateModels.py by simply running it and inputting the probability distributions
3. Make the adversary graphs. This is achieved by running createGraph.py against an adversary file generated above, for example:
   ```
   > python createGraph.py 2bandit50.adv
   ```
4. Make the reward comparison graph. This is achieved by running createRewardGraph.py against the three result files as such:
   ```
   > python createRewardGraph.py 2bandit50dtmc.res 2bandit50mdp.res
   2bandit50pomdp.res
   ```
   Here the files need to be in order of DTMC, MDP then POMDP.

6 Bibliography

E. W. Akin. A multi-armed bandit approach to following a Markov chain, 2017. URL `https://calhoun.nps.edu/handle/10945/55572`.

P. Blasco and D. Gündüz. Multi-access communications with energy harvesting: A multi-armed bandit model and the optimality of the myopic policy. *IEEE Journal on Selected Areas in Communications*, 33, 03 2015. doi: 10.1109/JSAC.2015.2391852.

E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, *Logics of Programs*, pages 52–71, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg. ISBN 978-3-540-39047-3.

E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-03270-8.

W. Dai, Y. Gai, B. Krishnamachari, and Q. Zhao. The non-bayesian restless multi-armed bandit: A case of near-logarithmic strict regret. pages 2940–2943, 09 2011. doi: 10.1109/ICASSP.2011.5946273.

V. Forejt, M. Kwiatkowska, and D. Parker. Pareto curves for probabilistic model checking. In S. Chakraborty and M. Mukund, editors, *Proc. 10th International Symposium on Automated Technology for Verification and Analysis (ATVA'12)*, volume 7561 of *LNCS*, pages 317–332. Springer, 2012.

J. Gittins, K. Glazebrook, and R. Weber. *Multi-Armed Bandit Allocation Indices, 2nd Edition*. Wiley, 02 2011. doi: 10.1002/9780470980033.ch8.

A. Hagberg, D. Schult, and M. Renieris. Pygraphviz, 2004. URL `http://pygraphviz.github.io`.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3): 90–95, 2007. doi: 10.1109/MCSE.2007.55.

K. Jamieson and R. Nowak. Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting. In *2014 48th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6, March 2014. doi: 10.1109/CISS.2014.6814096.

J. Kuhn and Y. Nazarathy. *Wireless Channel Selection with Restless Bandits*, pages 463–485. 03 2017. ISBN 978-3-319-47764-0. doi: 10.1007/978-3-319-47766-4_18.

M. Kwiatkowska. Model checking for probability and time: From theory to practice. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 351–360. IEEE Computer Society Press, 2003. Invited Paper.

M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for Markov decision processes. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 157–166. IEEE CS Press, 2006.

M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4): 40–45, 2009.

M. Kwiatkowska, G. Norman, and D. Parker. Advances and challenges of probabilistic model checking. In *Proc. 48th Annual Allerton Conference on Communication, Control and Computing*, pages 1691–1698. IEEE Press, 2010.

M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

J. Le Ny, M. Dahleh, and E. Feron. Multi-uav dynamic routing with partial observations using restless bandit allocation indices. In *2008 American Control Conference*, pages 4220–4225, June 2008. doi: 10.1109/ACC.2008.4587156.

H. Liu, K. Liu, and Q. Zhao. Learning in a changing world: Restless multiarmed bandit with unknown dynamics. *IEEE Transactions on Information Theory*, 59(3):1902–1916, March 2013. ISSN 0018-9448. doi: 10.1109/TIT.2012.2230215.

A. Mahajan and D. Teneketzis. *Multi-Armed Bandit Problems*, pages 121–151. Springer US, Boston, MA, 2008. ISBN 978-0-387-49819-5. doi: 10.1007/978-0-387-49819-5_6. URL `https://doi.org/10.1007/978-0-387-49819-5_6`.

G. Norman, D. Parker, and X. Zou. Verification and control of partially observable probabilistic systems. *Real-Time Systems*, 53(3):354–402, 2017.

H. Oldenkamp. Probabilistic model checking : a comparison of tools. 2007.

C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of optimal queueing network control. In *Proceedings of IEEE 9th Annual Conference on Structure in Complexity Theory*, pages 318–322, June 1994. doi: 10.1109/SCT.1994.315792.

K. Sigman. Discrete-time Markov chains: Stochastic processes in discrete time. 2008.

P. Whittle. Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability*, 25:287–298, 1988. ISSN 00219002. URL `http://www.jstor.org/stable/3214163`.