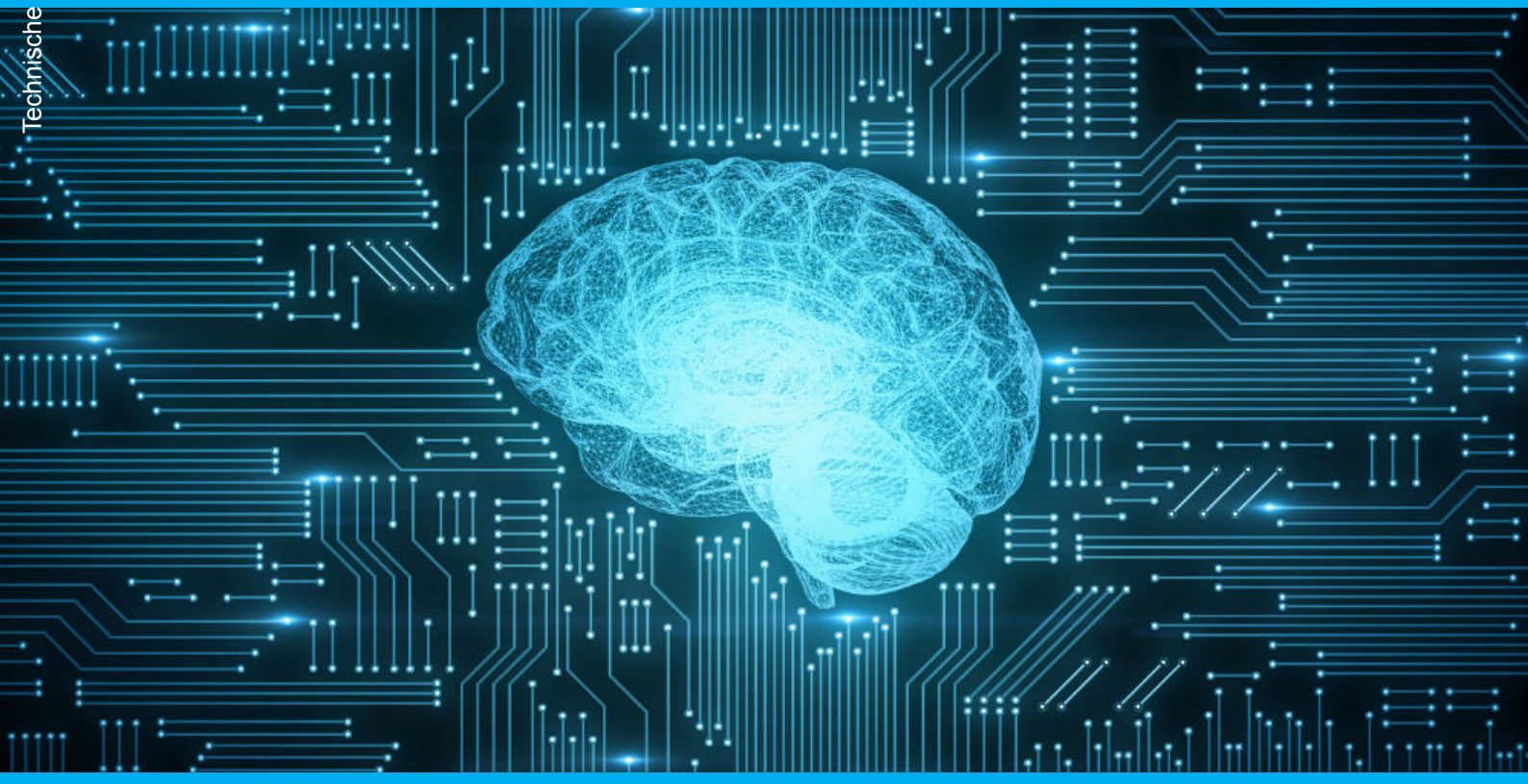
Reinforcement Learning in Block Markov Chains

P.G. (Pascal) Lagerweij

Technische Universiteit Delft



TU Delft

NAS

PVM2019-101

Reinforcement Learning in Block Markov Chains

by

P.G. (Pascal) Lagerweij

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday March 29, 2019 at 10:15 AM.

Student number:	4088301	
Faculty:	EEMCS	
Master program:	Electrical Engineering	
Project duration:	May 22, 2018 – March 29, 2019	
Thesis committee:	Prof. dr. ir. P.F.A. Van Mieghem	TU Delft
	Dr. ir. J. Sanders	TU Delft, daily supervisor
	Dr. M.T.J. Spaan	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.
Picture on cover taken from http://www.merl.com/research/artificial-intelligence.

TU Delft

Preface

Before you lies the thesis ”Reinforcement Learning in Block Markov Chains”. This thesis has been written to obtain the degree Master of Science at the Delft University of Technology (TU Delft). The project was conducted at the Network Architectures and Services (NAS) group, within the faculty of Electrical Engineering, Mathematics and Computer Science at Delft University of Technology. Under daily supervision of dr. ir. J. Sanders I was engaged in researching and writing this thesis from May 2018 to March 2019.

Delft, including the TU Delft, has been the place where I learned how to take care of myself, and where I started discovering who I am and what I want to be. It has been a long and enjoyable journey, starting from the first day at the TU Delft, but now the last day at the TU Delft is in sight. Here we are; a Bachelor, a year as treasurer in the board of the study association for electrical engineering student, the Electrotechnische Vereeniging (ETV), six months of studying at the Ecole Polytechnique Fédérale de Lausanne in Switzerland, and a master thesis later. We arrived at the final stepping stone to officially call myself an ”ingenieur”. This part of my journey through life is about to come to an end, and the next chapter is about to begin and I am looking forward to whatever life will bring me. Before we conclude this chapter of my life, I would like to acknowledge those who helped me get to the point where I am today.

First, Jaron, I honestly do not think I would have been where I am today without you. You always took the time for meetings and discussions and gave me a push to keep going forward when I needed it the most. I sincerely hope that your goals and ambitions become reality and I wish you all the best.

Prof. dr. ir. P.F.A. Van Mieghem and dr. M.T.J. Spaan for being part of my thesis committee.

Of course, I would like to thank all MSc. students in the graduation room for the chats and for making sure the coffee breaks were much more enjoyable. Furthermore, I would like to express my gratitude to the entire NAS group, for the nice lunches, and the discussions and joy we had on the Friday afternoon drinks.

I am grateful for all my fellow student who made my time at the university so much more enjoyable. I would like to explicitly mention my fellow board members of the ETV and my ”clubgenoten” of Virgiel.

Last of all, I want to thank my parents and brother, for all the support they gave to and belief they had in me.

Now, it is time to conclude this chapter of my life and start a new chapter, wherever it may take me!

P.G. (Pascal) Lagerweij
Delft, March 2019

Abstract

Nowadays, reinforcement learning algorithms on Markov decision processes (MDPs) face computational issues when the state space is large. To reduce this state space of a MDP several state aggregation, or clustering, methodologies have been applied. Recently, a new clustering algorithm has been proposed that is able to cluster states from a single block Markov chain. A block Markov chain is a Markov chain with blocks in its transition matrix that correspond to clusters. Our aim was to investigate the possible combination of state aggregation in reinforcement learning on MDPs with clustering of states on a block Markov chain. First, we investigated the clustering algorithm and its properties to see its performance with different parameters and trajectory length. We compared the statistical properties of a pure Markov chain and the mixed Markov chain generated by a MDP. Afterwards, we verified the performance of the clustering algorithm on this mixed Markov chain. We proposed the BMC–MDP model that is able to model cluster based MDPs. We proposed C-PSRL, an algorithm, that consists of a single clustering step, on this newly introduced model. We compared its performance with a naïve approach and concluded that this new combined approach of clustering and MDP solving on a reduced space is a viable approach that reduces the computational complexity significantly. This research opened up the possibilities of more complex algorithms with, for example, multiple clustering steps. Moreover, if we can extend this clustering algorithm to clustering based on a state and action trajectory, this may results in an increased clustering performance and thereby enhance the performance of this general approach of optimizing on a cluster based MDP.

Contents

Acronyms

BMC Block Markov Chain. 11–17, 19, 22, 25, 30, 37, 38, 41, 42, 46, 55

BMC–MDP Block Markov Chain Markov Decision Process. 37–39, 41, 42, 55

C-PSRL Cluster Posterior Sampling for Reinforcement Learning. vi, 39–41, 43–49, 51, 56

CA Clustering Algorithm. 15–17, 19–23, 31–34, 39, 40, 42–45

CIA Cluster Improvement Algorithm. 15–17, 42

MDP Markov Decision Process. 5–9, 19, 23, 34, 37–39, 41, 43–49, 53, 55, 56

OFU Optimism in the Face of Uncertainty. 8

POMDP Partially Observable Markov Decision Process. 38

PSRL Posterior Sampling for Reinforcement Learning. 9, 39, 41, 43–45, 49, 56

RL Reinforcement Learning. 8, 39–41, 44, 53, 56

ROMDP Rich-Observation Markov Decision Process. 37, 38

SCA Spectral Clustering Algorithm. 15, 16

SVD Singular Value Decomposition. 16

TVD Total Variation Distance. 26–28

UCB Upper Confidence Bounds. 4

UCRL Upper Confidence for Reinforcement Learning. 9

1

Introduction

In Reinforcement Learning (RL) an agent interacts with an unknown environment. The agent tries to maximize its return by performing actions. This action influences the reward and the state of the environment. The challenge therefore is to understand how an action will affect the future rewards. The dominating approach in the field is to model this environment using a Markov Decision Process (MDP) [1], [2]. A MDP is a mathematical model dating back to the 1950s [3],[4]. In RL the agent is uncertain about the dynamics of the environment at the start of the process and must therefore *explore* different strategies in order to gain information about the environment. Once the agent obtains an understanding of the environment the agent can *exploit* this knowledge to compute a good strategy. The performance is usually measured in terms of *cumulative regret*; the difference between the reward of a strategy and the reward of the optimal strategy. RL has been applied to various problems in for example robotics [5], healthcare, finance, energy, and transportation [6]. Fascinatingly, RL is even being used to create agents to play games such as Tetris [7] and Go [8]. Recently, agents have been created to play complicated video games such as Super Smash Bros. Melee [9] and Starcraft II [10].

In the past decade many approaches to find the optimal strategy for a MDP with RL have been proposed and analysed. With the improvements in computational power over recent years many new techniques have been proposed that leverage various underlying model assumptions. This led to a rise in the complexity of the problems for which a solution can be found in a reasonable time. However, even with this computational power, algorithms still face issues when the state space is large. The number of possibilities increases exponentially and that makes some of the more refined methods take too much processing power to make them suitable for day to day use. For example, various concrete reinforcement learning problems that model dynamical systems using a Markov chain with an unknown transition kernel still face this issue [2].

To make the issue even more pressing, in practice we often deal with environments with large observation spaces. For standard RL algorithms the regret grows with the size of the observation state space. Nonetheless, in several domains there is an underlying latent space that describes this observation space and its dynamics and rewards. This latent space is of lower dimension than the total observation state space. If it is possible to find this lower dimensional latent space we can reduce the learning complexity because we only have to optimise in this lower dimensional latent space. To give an example of a latent space, if a robot has multiple sensors we receive input from every sensor but all these inputs can, for example, be translated into a 2D map of the environment of this robot. The assumption that a lower dimensional, latent, space exists has been utilised in RL. As a matter of fact, for multi-armed bandit algorithms, a bandit algorithm is a RL algorithm on a MDP with one state, it is shown that learning such a (latent) finite set reduces the regret significantly [11], [12]. Furthermore, state aggregation through different methods has been widely studied for Markov Decision Processes, see e.g. [13]. To illustrate the theoretical complexity of finding a latent space, consider that finding the minimal ϵ-equivalent MDP for a MDP in its tabular form is NP-hard [14]. The ϵ-equivalent MDP is a MDP with a smaller state space and in this MDP states that have similar transition probabilities and rewards are clustered. For states to be clustered the transition probabilities and reward have to be within a norm of ϵ of each other. Nevertheless, two algorithms have been proposed recently that aggregate (cluster) states and learn on the latent MDP. [15] proposed an algorithm combining state aggregation

with UCRL, a reinforcement learning algorithm. The result reduce the computational complexity but it does not show any improvement in the regret. [16] introduced the Rich-Observation Markov Decision Process (ROMDP) model and propose an algorithm combining clustering within this ROMDP model with UCRL. They show that they can cluster and that this algorithm asymptotically matches the regret of learning on the latent MDP.

The inspiration for this Thesis comes from the research [17]. They showed that clustering of states is possible on a single trajectory generated by a Block Markov Chain (BMC). A BMC is a Markov chain with a block structure in its transition matrix. These blocks can be seen as clusters, and these clusters form a latent space of the Markov chain. Because a Markov chain describes the dynamics of a MDP our goal is to research if we can utilise this clustering methodology to learn the latent structure of a Markov chain and with that the latent structure of the MDP. By learning this latent structure we hope to efficiently find a policy.

1.1 | Research questions and outline

We formulated the idea of combining the work of clustering in BMCs by [17] with RL into learning objectives of this thesis. Our main objective is to investigate the performance of a two-step approach of clustering and optimization of a MDP with an underlying BMC compared to the performance of a naïve RL implementation on the same MDP. If we can learn a latent space effectively we can reduce the computational complexity of our agent because this effectively is a reduced state space. Within this thesis we solely focus on a method where you first cluster the large state space into a smaller cluster space. We assume that the behaviour of the states within a cluster act similar in terms of transition dynamics and reward. By clustering we effectively reduce the state (cluster) space of the problem. Afterwards we optimize for this MDP utilising the latent cluster space.

This thesis is a first study on combining this clustering methodology with optimization in a MDP. Our first aim is to understand MDPs and clustering in BMCs. We implement a BMC simulator and gain insights in clustering on a BMC trajectory. We analyse the compatibility of the clustering algorithm for application to a trajectory generated by a MDP. After the analysis on clustering on BMCs, we formulate a model for a cluster based MDP. We propose an algorithm that is able to combine these two methodologies and show results on this algorithm's performance. This methodology gave us the tools necessary to analyse this combined approach and will be described by six chapters:

- In Chapter 2 an introduction is given on Reinforcement Learning. This chapter covers bandit problems and Markov Decision Processes. For both frameworks several algorithms are described.
- In Chapter 3 the definition of a Block Markov Chain and the recently developed algorithm to cluster the states of this Block Markov Chain are described. The rest of this chapter is devoted to an analysis and an investigation of the properties of this algorithm. These results lay the foundation for combining this clustering algorithm in the Markov Decision Processes framework.
- In Chapter 4 the clustering methodology is combined with the Markov Decision Processes. We present a new model to combine a Block Markov Chain structure with a Markov Decision Process. Afterwards, we present an algorithm that successfully deals with the cluster based Markov Decision Process problem and show results of the performance of this algorithm.
- In Chapter 5 we present the conclusions of the Chapter 3 and 4. We also give a general conclusion for the main objective of this thesis.
- In Chapter 6 recommendations for further research are given.

2

Reinforcement Learning

Learning through interacting with our environment is probably one of the most fundamental ways of learning when we think about the nature of learning. Every human does this on a daily basis; for example, when growing up we learn to ride a bike with some guidance from our parents. But afterwards we learn to ride a bike without touching the handlebar. We do this through interaction with the environment, specifically the bike on the road. We take actions: we lean in different directions and see how the bike reacts. By leaning we learn how to ride a bike without even tugging the handlebar through repeated interaction. At some point we understand that if the bike starts falling to the right we have to take an action, namely leaning towards the left, in order to not fall to the ground. This example describes how learning through interaction is a fundamental method of learning and intelligence.

This chapter introduces Reinforcement Learning, a computational approach to learning from interaction, and we will describe models that are used as a learning framework. The objective is to find the optimal action in the different scenarios we encounter. We start this chapter with an introduction to Reinforcement Learning in Section 2.1. The first model we introduce is the bandit problem, this problem is presented in Section 2.2. In Section 2.3 multiple algorithms to find the optimal action strategy for the bandit problem are described. Afterwards in Section 2.4 we describe a more complex model that is the key model studied in this thesis: the Markov Decision Process model. We conclude this Chapter by describing algorithms that find the optimal actions in Markov Decision Processes in Section 2.5.

2.1 | Reinforcement Learning

Reinforcement learning is a computational approach to learning from repeated interaction with its environment. This kind of learning is also referred to as *sequential decision making under uncertainty*. Sequential decision making is a general term to describe a situation where the agent or decision maker does successive operations on an environment to find the optimal decision for that specific environment. A sequential decision problem is therefore a concatenation of decision problems, and each time an action should be taken while taking its effect on the future into account [18]. The objective of the agent is to map each situation to an optimal action to be taken. We use the term agent to describe the decision maker in this repeated learning process. The agent uses the information that is acquired to make the decision. Most of the material in this Chapter can be deducted from [2] or [1].

Generally speaking the objective of the agent, in this computational approach, is to maximize a numerical reward signal. The agent is not told which actions to take, but must discover which actions yield the most reward by trying them, hence the uncertainty. Therefore the agent has to repeatedly interact with the environment to find out the best action for every observation. In challenging environments, actions may not only affect the reward received at this time point but also influence the next observation, and through that, all subsequent rewards that are received.

In Figure 2.1 a general framework for reinforcement learning is shown. The agent interacts with the environment and based on the reward and observation it receives the agent determines the next action. The agent must be able to observe the situation through observation and must be able to take actions that affect this situation in order for a reinforcement learning problem to make sense. The agent should also have clear goals related to the state of the environment. Here the desired state of the environment

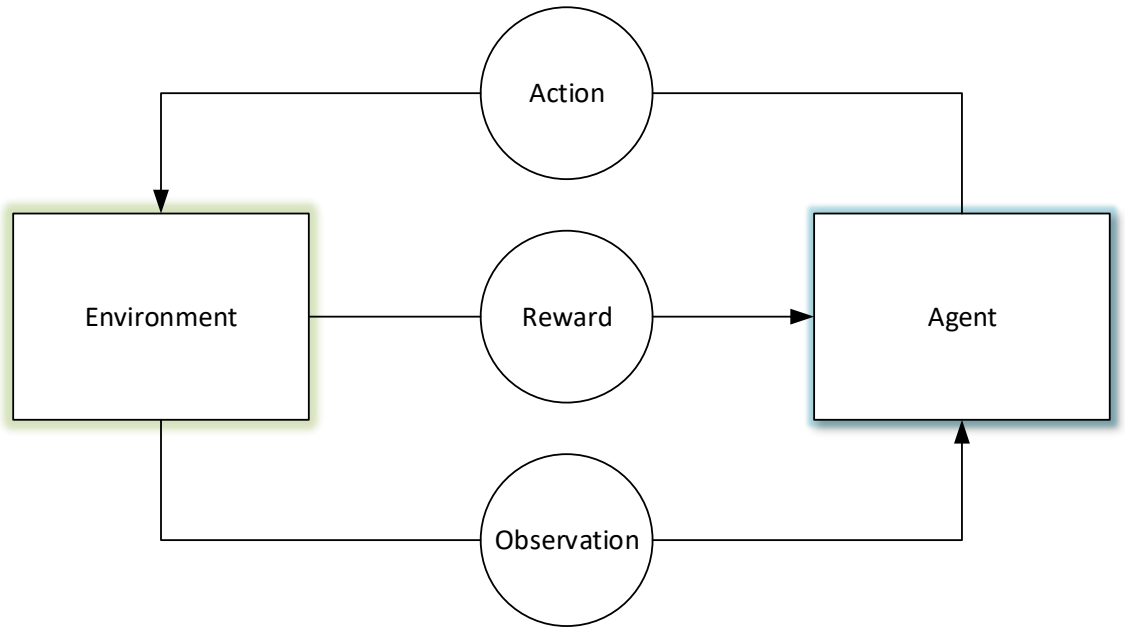


Figure 2.1: General framework of Reinforcement Learning. The agent interacts with the environment and based on the reward and observation it receives it determines the next action that is taken.

is modelled through the reward that is obtained. We will formalize this problem of finding the optimal action on incompletely-known Markov decision processes, the details will follow in Section 2.4. For now, we note that if a general problem can be described in such kind of framework it can be solved with a reinforcement learning method. From [2], *"As Markov decision processes are intended to include just these three aspects - sensation, action and goal - in their simplest form without trivializing any of them. Any method that is well suited to solving such problems we consider to be a reinforcement learning method."*

The major challenge in reinforcement learning is the trade-off between exploration and exploitation. The agent is uncertain about the environment and without the full knowledge of the environment the agent faces a conflict between exploring the environment and the objective to exploit the environment to increase her reward. The trade off between exploring and exploiting is worsened by the fact that the agent's actions influence the information gained. Thus in order for the agent to learn efficiently the right balance has to be struck between exploration and exploitation.

To summarise: in reinforcement learning we start with an interactive, goal-seeking agent and an environment. This reinforcement learning agent has explicit goals, can sense aspects of the environment, and can choose actions to influence the environment. It is usually assumed that the agent has to operate from the beginning despite uncertainty about the environment it faces. Hence reinforcement learning involves planning, the agent has to balance action selection for exploration and exploitation, as well as the question of how we model the environment and improve the estimate of this environment.

2.1.1 | Elements of Reinforcement Learning

Besides the agent and environment we can identify four main elements in a reinforcement learning system: a policy, a reward signal, a value function, and optionally, a model of the environment.

The *policy* defines the way the agent picks his action at a given time. This means that the policy is a mapping from a current state of the environment and time to the action taken at that time. The policy could be a look up table with a determined action for any situation but could also be stochastic, specifying probabilities for every action.

The *reward signal* is used to define the goal of the reinforcement learning problem. At every time step the environment sends the agent a single number, the reward. The agent's objective is to maximize the sum of those received rewards. The reward signal essentially determines what is a good

or bad event for the agent. In general, reward signals may be stochastic functions of the state of the environment and the action taken.

While the reward signal determines the immediate effect of any action, the *value function* takes the long time expectation into account. Roughly speaking, the value function determines the expected cumulative reward you can expect over time starting from picking the action in that specific state. As mentioned before, challenging environments may potentially have long lingering effect: even if the immediate reward of picking that action is low, the value function for that action may be high if subsequently the environment is often taken to states with a high reward.

The last element is the *model* of the environment. This is an optional parameter in a reinforcement learning system. Reinforcement learning systems that utilise a model are called model-based algorithms. These models are used for planning as we can use the estimated model to plan the current action. We do this with a model because we can take future situations into account before they occurred and in that way determine the current policy more effectively.

2.2 | A K-armed Bandit Problem

The first reinforcement learning problem we will now explore is the bandit problem. The bandit problem is a special case of the general reinforcement learning problem because it only consists of a single state. The bandit problem is a simplified problem and can be illustrated without the observation input as illustrated in Figure 2.2. Stochastic bandits are widely researched and one the earliest references was by [19] who proposed an algorithm that will be described in Section 2.3.3.

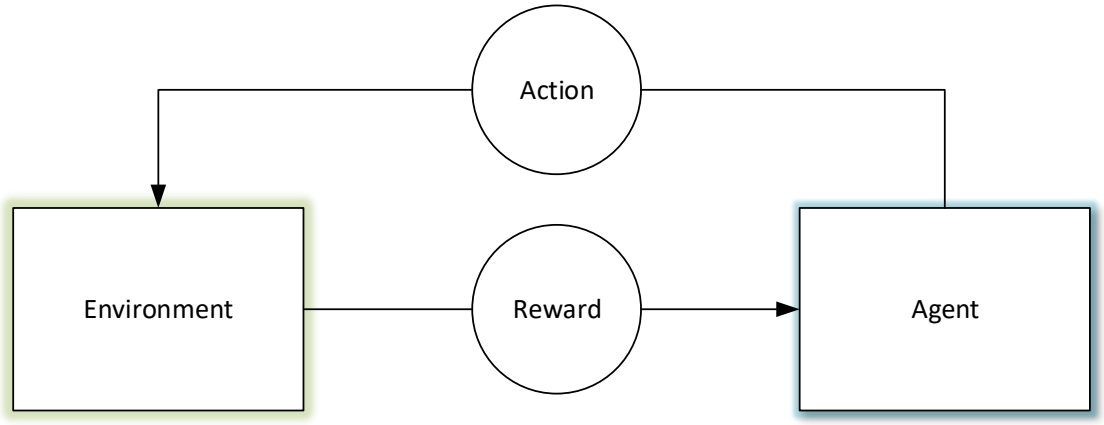


Figure 2.2: General framework of the bandit problem. The agent interacts with the environment and based on the reward it determines the next action.

In this section we will study the multi armed bandit problem that is also referred to as the stochastic bandit problem. The agent is repeatedly faced with a choice among K different options (actions). After every action the agent receives a reward and his objective is to maximize the expected total reward over a time period T. Mathematically the stochastic bandit problem consists of K independent probability distributions (arms) $\{D_1, D_2, \dots, D_K\}$ with expected values $\{\mu_1, \dots, \mu_K\}$. The agent does not know those K distributions in advance and the objective of the agent is to find out the optimal distribution (arm) to play at according to the performance evaluation function. In the situation described earlier the performance evaluation function is the sum over the received rewards from $t = 1, \dots, T$. At time step t the agent selects an arm (one of the K distributions): this choice represents the action $A_t \in \{1, \dots, K\}$ and the agent receives a reward $R_t \sim D_{A_t}$. The expected reward at that time point t is $\mathbb{E}[R_t] = \mathbb{E}[D_{A_t}] = \mu_{A_t}$ and depends on the arm A_t that is picked. The optimal strategy would be to pick the arm that has the largest expected reward, so we define the optimal action $A^* = \operatorname{argmax}_{j=\{1,\dots,K\}} \mu_j$ as choosing the arm with the highest expected reward $\mu^* = \max_{j=\{1,\dots,K\}} \mu_j$. The agent thus has a two sided goal of on the one hand finding out which arm gives the largest expected mean reward and on the other hand playing the arm that yields the highest reward. This illustrates exploration versus exploitation principle. A bandit algorithm specifies a policy on which arm A_t to select at time step t.

The mentioned value function in the introduction to reinforcement learning is identical to the reward signal in the bandit problem. This is because there is no long term planning involved as there is only a single state. Next we will demonstrate the way we measure the performance of a bandit algorithm. The performance of a bandit algorithm is measured with the regret. The regret is the difference between the reward received using the policy of the algorithm and the reward received using the optimal policy. The total expected regret at time step T is

$$\mathbb{E}[\text{Regret}]_T = T\mu^* - \sum_{t=1}^{T} \mu_{A_t}. \tag{2.1}$$

Here μ_{A_t} is the expected reward if the arm A_t picked at time instance t and μ^* is the optimal reward value. The difference between the expected reward value of the optimal arm and the arm picked will be defined as the immediate regret for action j by $\Delta_j = \mu^* - \mu_j$. Hence $\mathbb{E}[\text{Regret}]_T = \sum_{t=1}^{T} \Delta_{A_t}$. The bandit problem was formally restated in a short paper by [20] and here the notion of regret was introduced.

Next we will illustrate these mathematical definitions with an example to give insight in what kind of problem this multi-armed bandit problem describes.

Example:
You, as the agent, stand in a casino and face the choice of $K = 3$ slot machines to play on. The slot machines have different expected rewards $\{\mu_1, \mu_2, \mu_3\}$ as illustrated in Figure 2.3. You have no knowledge of the three different slot machines and want to find the optimal action (slot machine) $A_t \in \{1, 2, 3\} \forall t \in \{1, \dots, T\}$ for you to play on. The optimal action is to play on the slot machine that gives the best reward so the slot machine with the highest expected reward μ^*. This is optimal because playing on this optimal slot machine gives the best expected reward and an immediate regret of zero for that time step.

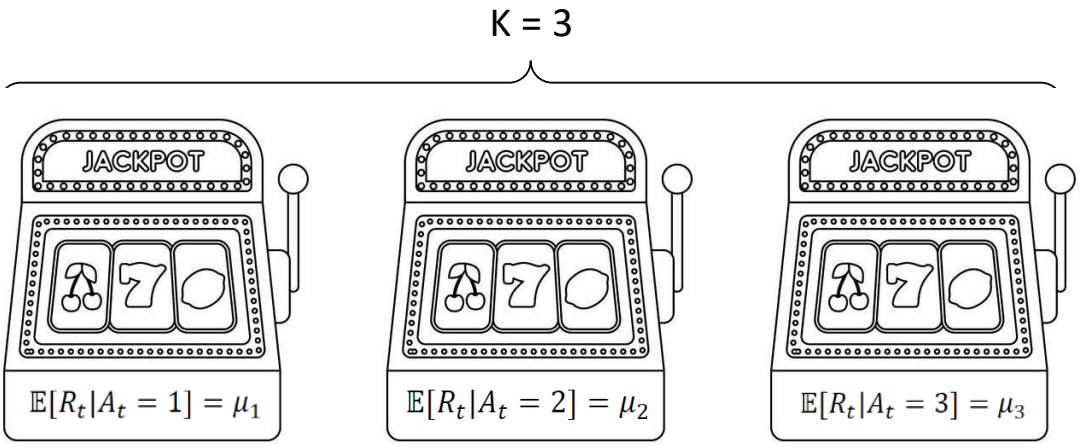


Figure 2.3: Illustration of a 3-armed bandit problem.

The performance of a policy is measured by the regret of that algorithm.The general performance of a reinforcement learning algorithm is commonly analysed by bounding the regret. A lower and upper bound on the regret, after a certain amount of time steps, gives an indication of how good an algorithm is. The first asymptotic lower bound on any bandit algorithm is derived by [21]. We know that an algorithm that follows this lower bound is guaranteed optimal in that sense. For an extensive study on bandit problems and algorithms we refer the reader to [22] or [23].

2.3 | Algorithms for Bandit problems

In this section we describe three algorithms that are used to find a policy for the K-armed bandit problem.

2.3.1 | The epsilon-greedy algorithm

In the K-armed bandit problem we have to select an action at every time step. To select this action we use the estimated reward mean of the K different actions and we do this by keeping track of the estimated mean $\hat{\mu}_t(j)$ at time step t for all actions $j \in \{1, \dots, K\}$. We also measure how certain we are of this estimate and we do this by keeping track of the number of times this action (arm) is played $N_t(j)$.

The greedy algorithm is initialised by playing every action once and setting the estimated reward $\mu_K(j) = R_j$ and $N_K(j) = 1$ for all actions $j \in \{1, \dots, K\}$. The next action A_t at $t = K + 1$ is picked by looking at the current estimated reward means and select the one that is the highest, i.e., $A_t = \operatorname{argmax}_{j=1,\dots,K} \hat{\mu}_{t-1}(j)$. We play this action and receive reward R_t. We then update these two quantities for $j = A_t$ using the received reward R_t iteratively:

$$\begin{aligned} \hat{\mu}_t(j) &= \hat{\mu}_{t-1}(j) + \frac{R_t - \hat{\mu}_{t-1}(j)}{N_{t-1}(j)}\,, \\ N_t(j) &= N_{t-1}(j) + 1\,. \end{aligned} \tag{2.2}$$

For the other actions $j \in \{1, \dots, K\} \neq A_t$ the estimate and the number of trials does not change and are updated by $\hat{\mu}_{t+1}(j) = \hat{\mu}_t(j)$ and $N_{t+1}(j) = N_t(j)$.

The estimator in (2.2) is not necessarily the best. Other estimators could be used but for didactic reasons we stick with this definition and focus on explaining how these estimates can be used to select actions.

The algorithm described above is *greedy* in the sense that it always picks the action that has the highest estimated mean at that time. This kind of greedy approach does not favour exploration of the other options and that is why a variant, the ϵ-greedy algorithm, was introduced. The ϵ-greedy algorithm is similar to the greedy algorithm, but it favours exploration, it selects a random action with probability ϵ. The ϵ-greedy algorithm therefore selects the arm with the highest estimated mean with probability $1 - \epsilon$, and it chooses a random arm with probability ϵ. Hence the probability of selecting arm A_t at time t is given by:

$$\mathbb{P}(A_t = a) = \begin{cases} 1 - \epsilon + \epsilon/K & \text{for } a = \operatorname{argmax}_{j=1,\dots,K} \hat{\mu}_{t-1}(j)\,, \\ \epsilon/K & \text{for } a \neq \operatorname{argmax}_{j=1,\dots,K} \hat{\mu}_{t-1}(j)\,. \end{cases}$$

If there are multiple actions with the same estimated mean we select any of those actions with a policy. As long as we use the same policy in all scenarios where there are multiple action this does not influence the performance of the algorithm.

If the reward would have zero variance the greedy algorithm would always outperform the ϵ-greedy algorithm. This is because all the estimates are exact after the initial stage of selecting every action once. However, if the variance is non-zero more exploration is needed to find the optimal action and the ϵ-greedy algorithm will outperform the greedy algorithm in most scenarios. In practice you want to explore more in the beginning and less later on as the estimates become more accurate, so often a varying ϵ is used. For example in [24, Theorem 3] they set ϵ proportional to $1/t$ and this ensures that ϵ decreases fast enough so that the regret is close to the optimal regret but slow enough that the selected action converges to the optimum arm.

2.3.2 | Upper Confidence Bounds algorithms

The next family of bandit algorithms we discuss is the Upper Confidence Bounds (UCB) class of algorithms. This class of algorithms was proposed in [24]. The ϵ-greedy algorithm forces another action to be tried but does that without preferences. The class of UCB algorithms makes this exploration choice based on the potential of every action to be optimal. It does so by looking at the difference between the estimates of every action with the maximum estimate while also taking the uncertainty in those estimates into account.

The UCB algorithms maintain the number of times every arm has been played ($N_t(j)$) as well as the estimated mean ($\hat{\mu}_t(j)$) of all arms $j = 1, \dots, K$, similar to the greedy algorithms. All arms are played once and afterwards the algorithm picks the next action A_t based on the following equation:

$$A_t = \operatorname*{argmax}_{j=1,\dots,k} \Big(\hat{\mu}_{t-1}(j) + \sqrt{\frac{c \ln t}{N_{t-1}(j)}} \Big)\,. \tag{2.3}$$

The number $c > 0$ is a constant that controls the degree of exploration, for UCB1 this $c = 2$. After every time step the number of times the arm is played and the estimated mean are updated in the same way as in the greedy algorithms, see (2.2).

The potential of every action to be optimal is mathematically nested in the right-hand side of (2.3). The square root is a measure of the uncertainty in the estimate. The quantity that is used is thus a sort of upper bound on the possible reward mean for every action. This is where the name Upper Confidence Bound stems from.

The UCB family of algorithms is one of the benchmark bandit algorithms because of its known bounds on the regret. For UCB1, if the reward is bounded in $[0, 1]$ the maximum expected regret after T time steps is given by [24, Theorem 1].

Theorem 1 (Auer, Cesa–Bianchi, Fischer, 2002)**.** *For all $K > 1$, if policy UCB1 is run on K machines having arbitrary reward distributions $D_1, \dots, D_k$ with support in $[0, 1]$, then its expected regret after any number T of plays is at most*

$$\mathbb{E}[\textit{Regret}]_T \leq 8 \sum_{i:\mu_i<\mu^*} \left(\frac{\ln T}{\Delta_i}\right) + \left(1 + \frac{\pi^2}{3}\right)\left(\sum_{j=1}^{K} \Delta_j\right).$$

Here $\Delta_j := \mu^* - \mu_j$ is the immediate regret of playing action j, where $\mu_1, \dots, \mu_K$ are the expected values of $D_1, \dots, D_k$.

2.3.3 | Thompson Sampling algorithm

Thompson Sampling (TS) was originally proposed in [19] and was the first bandit algorithm. The algorithm was independently rediscovered in [25], [26] and [27]. An introduction containing various examples can be found in [28]. The idea behind Thompson Sampling (TS) is to pick the arm that is likely to have the largest expected mean when we also take the number of times the arm has been played into account. While this idea is similar to the upper confidence bound methods, the approach of TS instead uses a prior distribution. At every point in time the algorithm samples an environment using this prior distribution and acts according to the optimal action in that sampled environment. This is why TS is also referred to as posterior sampling.

We will illustrate this concept with a TS algorithm that uses a Beta(α, β) distribution as the prior distribution. The Beta distribution is used to solve Bernoulli bandits. A Bernoulli bandit is a bandit with a Bernoulli distributed reward. A Beta distribution is used because of its conjugacy properties. Note that because the expected value of a Beta(α, β) distribution is given by $\alpha/(\alpha + \beta)$, the distribution becomes more concentrated as $\alpha + \beta$ grows. For the Bernoulli distributed reward the α models the successes (reward = 1), by $S(j)$ we denote the number of successes of arm j, and the β the failed rolls (reward = 0), we denote the number of fails of arm j by $F(j)$. The estimated mean of this Beta$(S(j), F(j))$ distribution $\mathbb{E}(\text{Beta}(S(j), F(j)) = S(j)/(S(j) + F(j)) = \hat{\mu}_j$ then corresponds to the estimated mean of the arm j. This estimate will convergence to the true mean as the number of samples increases. In this algorithm we initialize both these $S(j)$ and $F(j)$ with a 0 for every arm $j = 1, \dots, K$ because the Beta$(1, 1)$ distribution is the uniform distribution. Given the initialisation of $S_0(j) = 0, F_0(j) = 0 \quad$ for all $\quad j \in \{1, \dots, K\}$, we pick the next A_t at time t according to

$$A_t = \underset{j=1,\dots,K}{\operatorname{argmax}} \text{Beta}(S_{t-1}(j) + 1, F_{t-1}(j) + 1).$$

After playing arm A_t, we receive the reward R_t. We then update the S and F parameters according to

$$(S_t(A_t), F_t(A_t)) = (S_{t-1}(A_t) + R_t, F_{t-1}(A_t) + 1 - R_t). \tag{2.4}$$

The first TS asymptotic regret finite time analysis for Bernoulli bandits is given by [29, Theorem 1]. We reproduce the results here for your convenience:

Theorem 2 (Kaufmann, Korda, Munos, 2012)**.** *For every $\epsilon > 0$ there exist a problem-dependent constant $C(\epsilon, \mu_1, \dots, \mu_k)$ such that the regret of Thompson Sampling satisfies*

$$\mathbb{E}[\textit{Regret}]_T \leq (1 + \epsilon) \sum_{a\in 1,\dots,K:\mu_a\neq\mu^*} \frac{\Delta_a(\ln(T) + \ln\ln(T))}{K(\mu_a, \mu^*)} + C(\epsilon, \mu_1, \dots, \mu_k).$$

Here $K(p,q)$ is the Kullback–Leibler divergence, which is given by

$$K(p,q) \triangleq p \ln \frac{p}{q} + (1-p) \ln \frac{1-p}{1-q}.$$

The Beta distribution prior can be applied to general reward functions as long as the rewards are bounded on the interval $[0,1]$. The regret bound no longer holds for this more general setting as this bound is derived for a Bernoulli bandit. For general TS any prior distribution could be selected and for every arm we update the parameters that are needed to draw a sample from this distribution using the received rewards.

2.4 | Markov Decision Process

The second model that we introduce is the Markov Decision Process (MDP). A MDP can be seen as an extension of the multi-armed bandit problem. Unlike bandit problems, which optimise for immediate reward, a MDP factors in the long term consequences. An action taken at the current decision epoch can influence the system at future decision epochs and this requires planning over multiple epochs [30]. A MDP is a mathematical model dating back to the 1950s [3],[4]. The information in this section is quite general and can for instance be found in [31] or [32]. A MDP utilises the Markov property and we will start with general Markov chain properties in Section 2.4.1. Afterwards we will define the MDP model in Section 2.4.2, and in Section 2.4.3 we will conclude with a description of an algorithm that is able to find the optimal strategy in a MDP when all parameters of the problem are known.

2.4.1 | Markov chain

In a MDP the environment is modelled with states. This state represents the observation that the agent makes and on which the agent bases her action taken on, in combination with the received rewards. The dynamics that govern how the system evolves from the current state to the next state is assumed to satisfy the Markov property. Background information on Markov chains can be found in e.g. [31] and [33].

A discrete Markov chain is a sequence of random variables $\{S_0, S_1, S_2, \dots\}$ in which the variable S_t take their values in a discrete set $\mathcal{S}$ also called the *state space*. We call this sequence $\{S_t, t = 0,1,2,\dots\}$ a Markov chain if the transition probability from state to state satisfy the Markov property. The Markov property dictates that the history of the sequence is all factored in the current state. Therefore the transition probability only depends on the current state, i.e.,

$$\mathbb{P}(S_{t+1} = i_{t+1} \mid S_t = i_t, S_{t-1} = i_{t-1}, \dots, S_0 = i_0) = \mathbb{P}(S_{t+1} = i_{t+1} \mid S_t = i_t) \tag{2.5}$$

for all $i_0, i_1, \dots, i_{t+1} \in \mathcal{S}$.

If the transition probability $\mathbb{P}\{S_{t+1} = i_{t+1} \mid S_t = i_t\}$ does not depend on the time step t we call the Markov chain homogeneous. The transition probability from state x to y can then be expressed using a matrix P with entries

$$P_{x,y} = \mathbb{P}(S_{t+1} = y \mid S_t = x) \quad \text{for all} \quad x, y \in \mathcal{S}. \tag{2.6}$$

Because this matrix contains transition probabilities the sum of the every row is one; $\sum_{y \in \mathcal{S}} P_{x,y} = 1$ for all $x \in \mathcal{S}$.

2.4.2 | General structure

Now that we have seen the basics of a Markov chain and the transition matrix we can formulate the model of a MDP. In Figure 2.4 a graphical representation of the dynamics of a MDP is shown. At every time point t we have the state of the environment S_t. The agent selects action A_t and we receive the reward R_t and the environment progresses to next state S_{t+1}. At this next time point $t+1$ this process repeats itself. The arrows in Figure 2.4 illustrate dependencies in this process. The objective of the agent is to optimize a performance evaluation function, for example the sum of the received rewards $\sum_{t=1}^{T} R_t$.

Now that we have illustrated the dynamics we formulate the MDP model exactly. The dependencies in Figure 2.4 will also become clear from this. A finite MDP can be described with the Markov tuple $M = \langle \mathcal{S}, \mathcal{A}, R^M, P^M \rangle$. All n possible states are represented in the state space $\mathcal{S} = \{1, \dots, n\}$. Similarly all l possible actions are represented in the action space $\mathcal{A} = \{1, \dots, l\}$. At each time step $t = 0, \dots, T$

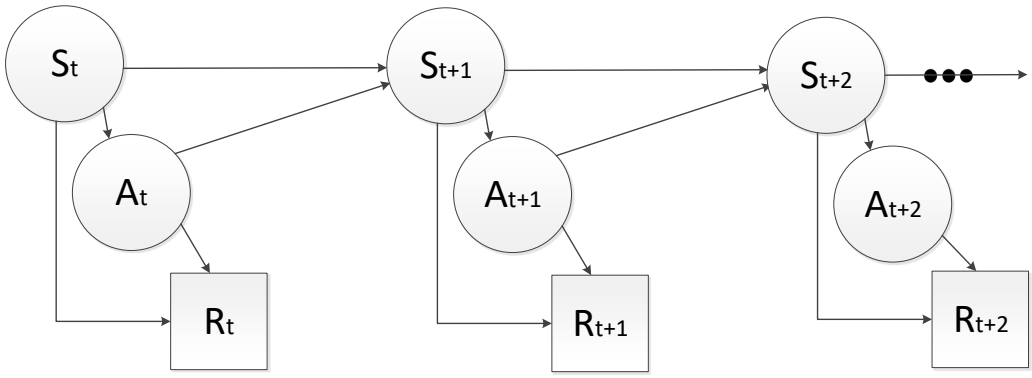


Figure 2.4: Graphical model of a Markov Decision Process.

the agent observes state $S_t \in \mathcal{S}$ and the agent selects action $A_t \in \mathcal{A}$. Afterwards the agent receives a reward R_t which is drawn from a distribution $R^M(S_t, A_t)$ which depends on the current state and action. The last model element we describe is the transition from state S_t to the new state S_{t+1}. This next state depends on the current state and the action and is described in the P^M matrices of the Markov tuple M. This P^M contains l, number of actions, Markov transition matrices, as seen in (2.6). So $P^M(S_t, A_t)$ is an array of length n that contains the transition probability to all n states given the current state and action.

In this definition we assume that the MDP is homogeneous, which means that the transition probabilities and rewards are independent of the decision epoch t, which means that they only depend on the current state and action. We also assume that the action space is finite and equivalent for all states.

The combination of a Markov Decision Process and the performance evaluation function is referred to as a Markov Decision Problem. The agent can solve such problem by using a policy. A policy is a sequence of decision rules $\{\pi_1, \pi_2, \dots, \pi_t, \dots\}$ where π_t is a decision rule at time epoch t. The decision rule at decision epoch t may depend on all available information in the system at that decision epoch. This means that the policy at time point T can depend on all previous states, actions, and rewards from $t = 1, 2, \dots T-1$. We denote a realisation of such a history by

$$\mathcal{H}_t \triangleq (S_1, A_1, R_1 \dots, S_{t-1}, A_{t-1}, R_{t-1}, S_t)\,. \tag{2.7}$$

The decision rule for decision epoch t can be seen as a function that contains the probability of an action to be taken given the history $\mathcal{H}_t$ and current state S_t , i.e.,

$$\pi_t(\mathcal{H}_t) = \mathbb{P}(A_t = a \mid \mathcal{H}_t) \quad \text{for all} \quad a \in \mathcal{A}\,. \tag{2.8}$$

A policy is said to be memoryless if the decision rule at time point t only depends on the current state S_t and not on the history $\mathcal{H}_t$. If a policy is independent of the decision epoch t the decision policy is called stationary. If this decision rule yields an action with probability one for every state, the policy is said to be deterministic. We denote a memoryless and deterministic policy by μ. This policy μ is thus a function mapping each state $S \in \mathcal{S}$ and decision epoch $t = 1, \dots, T$ to an action $A \in \mathcal{A}$, this can be formulated as $A_t = \mu_t(S_t)$.

Just as with the bandit problem we want to compare a policy, or algorithm, with the optimal algorithm. To compare a policy with the optimal policy we use the concept of regret. To define the regret for a MDP algorithm we have to look beyond the immediate reward, and this differs from the bandit problem. We first introduce the value function. The value function assigns a value to a state–action pair while taking the future of the system into account. For a MDP M and policy μ with finite horizon T we define the value function at step t by

$$V^M_{\mu,T}(s) \triangleq \mathbb{E}^M_\mu\Big[\sum_{j=t}^{T-1} \bar{r}(S_j, \mu(S_j)) \mid S_t = s\Big] + R_T(S_T)\,. \tag{2.9}$$

Here $\bar{r}(s,a) = \mathbb{E}[R \mid R \sim R^M(s,a)]$. This expectation indicates that the state $S_t = s$ and the actions for the entire finite time horizon are picked according to the policy μ. This gives the estimated total cumulative reward of policy μ.

A iterative form that only depends on the value function at the next time step is given by

$$V^M_{\mu,T}(s) = \sum_{s' \in \mathcal{S}} P^M_{s',s,\mu(s)} \mathbb{E}[R^M(s,\mu(s))] + V^M_{\mu,T}(s')\,. \tag{2.10}$$

The value function determines whether a policy μ is optimal for the MDP M. The policy μ^M is optimal if $\mu^M = \text{argmax}_{\mu'} V^M_{\mu',t}(s)$ for all $s \in \mathcal{S}$ and $t = 1, \ldots T$. We will denote by μ^M the policy that is optimal for M. This value function and the maximum arguments are also referred to as the Bellman equations or optimality equations.

Now that we have defined the optimal policy and the value function, we can define the regret. Similar to the bandit case we compare the algorithm for a MDP with the optimal algorithm μ^M. We define the regret incurred by a policy μ up to time T to be

$$\text{Regret}(T,\mu,M) \triangleq \sum_{k=1}^{T} \Delta_k = \sum_{k=1}^{T} \left(V^M_{\mu^M,T}(s_k) - V^M_{\mu,T}(s_k) \right). \tag{2.11}$$

Here Δ_k denotes the immediate regret over the k-th time step. This $\Delta_k = V^M_{\mu^M,T}(s_k) - V^M_{\mu,T}(s_k)$ is defined with respect to the MDP M and gives the difference between the current policy and the optimal policy.

This regret in (2.11) is complicated compared to the regret of the bandit problem in (2.1). This is due to the facts that the bandit problem only considers the immediate reward and thus there is no influence from the action on the dynamics of the system.

Example:
In Figure 2.5 we sketch a shop that can be modelled using a MDP. Imagine that you are a burglar breaking into a shop. Now this specific shop has a 4x3 layout, and we can model every square in this layout as a state. The action you take determines the direction you walk towards. This illustrates that there is a different transition matrix for every action. You start in the starting state and can take the action walk left for an immediate reward of 1. The objective however is to optimize the cumulative reward and if you take a look at the general layout of the shop it is clear that this is not the optimal strategy. The optimal course would be to take a walk in the shop, here the cost of the walk is modelled with a negative reward of -0.1 for every time step, to ultimately find the location with the 10 euro. However there is also the risk of getting caught which is modelled here with a negative reward of -10. If the entire shop is known, including transition matrices and rewards, the burglar is able to determine the optimal strategy. A method to calculate this optimal strategy will be demonstrated in the next section. The difficulty comes when the environment is not known a priori and you have to explore the environment. Here the trade off between exploration and exploitation comes into play. For example if you do not allow for exploration, an algorithm will be short sighted and will always pick the option walk left for an immediate reward of 1.

2.4.3 | Finding the optimal policy

To find the optimal policy for the Markov tuple M there are several methods including value iteration and policy iteration. For solving a finite-horizon discrete MDP one efficient method is *backward induction*, that is synonymous to dynamic programming. In literature however, the expression dynamic programming often refers to all methods for solving sequential decision processes. Another option to find the optimal policy, and for a stochastic problem the only option, would be to enumerate and evaluate all policies. Enumerating all policies is a computational demanding option.

We now present the backward induction algorithm that can be used to find the optimal policy in finite-horizon discrete MDPs. The backward induction algorithm is described in Algorithm 1. This algorithm uses the value function to determine the optimal policy and does this, as the name suggest, starting at the end of the time horizon $t = T$ and then works backwards to the starting time point $t = 1$. We assume here that the entire Markov tuple M is known. In this algorithm we utilise that the value function only depends on the current state and is deterministic. Here it is assumed that the MDP has an immediate reward for ending in a certain state hence the notation $R_T(s)$, if there is no such structure you can set the value function V^*_T to be 0 for all states. The backward induction algorithm then computes the value function for each previous epoch using the value function for each next epoch. The calculation

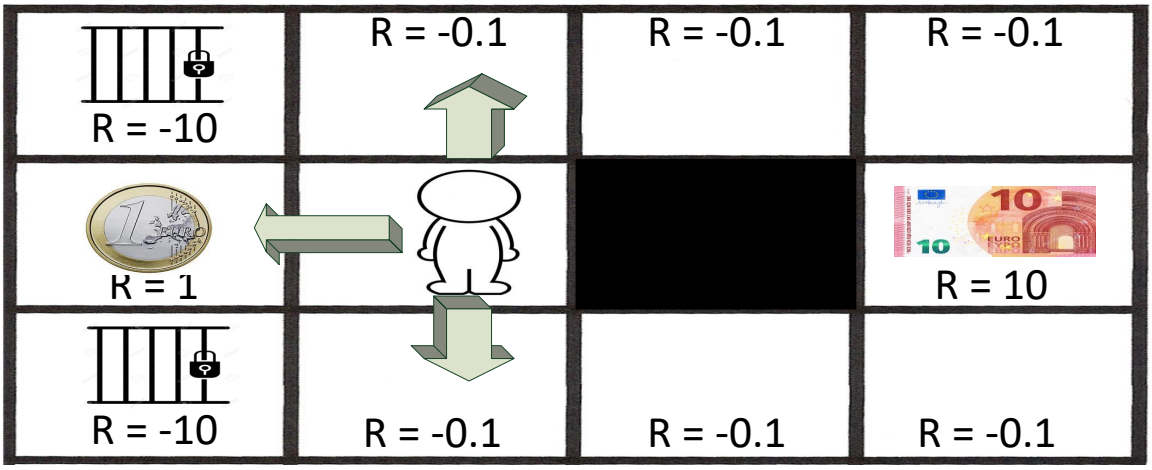


Figure 2.5: Illustration of a MDP problem.

is iterated until the start of the horizon. After $T-1$ iterations the optimal policy is found for every state at every decision epoch.

Algorithm 1: Backward induction

Input : Markov tuple M, walk length T
Output: Optimal policy $\mu_T^M(S_t = s, t)$ for all $s \in \mathcal{S}$

Set $V_T(s) = R_T(s)$ for all $s \in \mathcal{S}$
for $t = T-1, \dots, 1$ **do**
  for $s \in \mathcal{S}$ **do**
    $V_t(s) = \max_{a \in \mathcal{A}} \{\mathbb{E}[R_t(S_t = s, A_t = a)] + \sum_{j \in \mathcal{S}} \mathbb{P}_t(S_{t+1} = j \mid S_t = s, A_t = a) V_{t+1}(j)\}$
    $\mu_T^M(S_t = s, t) = \operatorname{argmax}_{a \in \mathcal{A}} \{\mathbb{E}[R_t(S_t = s, A_t = a)] + \sum_{j \in \mathcal{S}} \mathbb{P}_t(S_{t+1} = j \mid S_t = s, A_t = a) V_{t+1}(j)\}$
  end
end

The backward induction algorithm is a general algorithm where the reward and transition probabilities could be depended on the decision epoch. If the rewards and transition probabilities are homogeneous the resulting optimal policy depends only on the current state and is independent of the time instance, this results in a deterministic policy.

2.5 | Reinforcement Learning on a MDP

Recall that in RL the environment is unknown and has to be estimated by the agent. In this Section we consider the environment, so the reward R^M and transition probabilities P^M to be unknown to the agent. Before we introduce the learning algorithms, we will first expand the definition of an MDP to include episodes. A finite horizon MDP $M^* = (\mathcal{S}, \mathcal{A}, R^M, P^M, H, \rho)$ is defined as above but every H time steps the state will reset according to the initial distribution ρ. H is called the horizon of the MDP. The agent will interact repeatedly with the environment over H time steps which we call an episode.

We introduce the Q-value function, which takes these episodes into account. The policy μ is a mapping from state $s \in \mathcal{S}$ and period $h = 1, \dots, H$ to action $a \in \mathcal{A}$. The value function for each period h is

$$Q_{\mu,h}^{M^*}(s,a) \triangleq \mathbb{E}_{M^*,\mu}\left[\sum_{j=h}^{H} \tilde{r}^M(s_j, a_j) \mid s_h = s, a_h = a\right]. \tag{2.12}$$

Here the expectation parameter implies that the actions are chosen according to the policy μ. From

this the value function because $V^M_{\mu,h}(s) \triangleq Q^M_{\mu,h}(s, \mu(s,h))$. The regret of an algorithm is still as in (2.11) but with the value function for episodes.

Here we also denote the policy μ^M to be the optimal policy for the MDP M. This means that $\mu^M = \text{argmax}_\mu V^{M^*}_{\mu,h}(s)$ for all $s \in \mathcal{S}$ and $h = 1, \dots, H$. Now that we defined episodes we will introduce two model based algorithm classes.

2.5.1 | Optimism in the Face of Uncertainty

Various reinforcement learning algorithms, such as UCRL2 [34] and REGAL [35] learn the parameters of MDPs using the Optimism in the Face of Uncertainty (OFU) principle. This principle has driven the progress in the field of reinforcement learning. The approach is driven by assigning an optimistic bonus to poorly understood states and actions similar to the UCB for the bandit problem. This is done to stimulate exploration in a reinforcement learning algorithm. The agent selects the actions with respect to this optimistic model of the environment. When the agent resolves its uncertainty by having executed a state action pair a sufficient number of times, the effect of this optimistic approach is reduced and the behaviour of the agent approaches the optimal behaviour. The optimistic principle is to [36]: *"Select the policy which would obtain the best possible rewards in the best plausible environment."* In Figure 2.6 we illustrate the OFU principle. To find the optimal strategy we estimate the unknown model parameters P^M and R^M. On both these estimates we can formally define a "confidence ball", the set that describes this confidence ball for the reward is denoted by $\mathcal{R}$. For the transition tensor this set is denoted $\mathcal{P}$. The function that generates these confidence sets we skip here for not overly complicating our exposition. When episodes pass, these confidence balls shrink and with that the set of possible parameters. If we are able to guarantee that the confidence set contains the real parameter the estimates become more accurate over time. Such an optimistic model-based RL algorithm is illustrated in Algorithm 2. The key

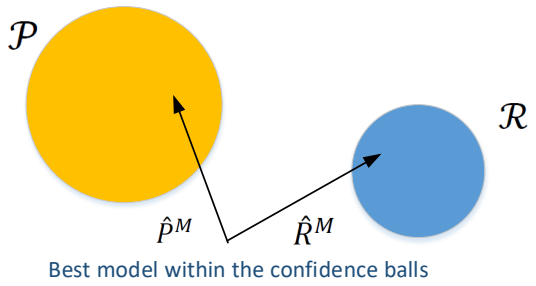


Figure 2.6: Graphical representation of the OFU principle.

step in this algorithm is the way you generate the confidence set (line 2 in Algorithm 2) and execute the optimistic planning (line 3 in Algorithm 2). The confidence sets have to contain the true MDP with high probability and also concentrate to a smaller set as fast as possible given the data. Next to the generation of the confidence set another key part of this algorithm is to plan optimistic over all policies μ and all MDPs $M \in \mathcal{M}_k$. A good approach to do this is extended value iteration[37], it presents a natural approach to optimistic planning. The algorithm uses $R^M_k(s,a,h) \in \max \mathcal{R}_k(s,a,h)$ for each (s,a,h). The question is how to assign optimistic transitions $P^M_k(s,a,h) \in \mathcal{P}_k(s,a,h)$. Extended value iteration has shown to do this with the increase of computational cost by a factor of $|\mathcal{S}| = n$ compared to solving a single MDP [34].

The main reason these types of algorithms are researched is that the there are regret bounds for these kind of learning algorithm. The key difficulty in bounding this regret is that this regret depends on the value of the optimal policy $V^M_{\mu^M,1}$. This value is unknown to the learning algorithm at the start. For example, Upper Confidence for Reinforcement Learning (UCRL) generates confidence sets for both reward and transition independently and takes the set $\mathcal{M}_k$ to be the set of all MDPs whose rewards and transitions lie within these confidence sets. The empirical mean will be a reasonable approximation to the unknown means of both the transition and reward. It has been proven that the true MDP will lie within the confidence sets $\mathcal{M}_k$ for all episodes k [36]. This makes UCRL a algorithm which produces efficient confidence sets. As a result of those efficient confidence sets, which contain the real MDP, UCRL will converge to the optimal policy.

Algorithm 2: Optimistic model-based RL

Input: Confidence set generator for MDPs Φ

for *episode* $k = 1, 2, \ldots$ **do**
  form confidence set $\mathcal{M}_k \sim \Phi(\cdot \mid \mathcal{H}_{k1})$
  compute $\mu_k \in \operatorname{argmax}_{\mu, M \in \mathcal{M}_k} V_{\mu,1}^{M}$
  for *time* $h = 1, 2, \ldots, H$ **do**
    take action $a_{kh} = \mu_k(s_{kh}, h)$
    observe r_{kh} and s_{kh+1}
    update $\mathcal{H}_{kh} = \mathcal{H}_{kh} \cup (a_{kh}, r_{kh}, s_{kh+1})$
  end
end

2.5.2 | Posterior Sampling for Reinforcement Learning

Besides UCRL, which can be compared to the UCB bandit algorithm, there is also an algorithm with a similar structure as the Thompson sampling approach for bandits. This algorithms learns the MDP using posterior or Thompson sampling and was originally proposed in [38] as *Bayesian Dynamic Programming*. It was later renamed to Posterior Sampling for Reinforcement Learning (PSRL) and proven to be an efficient algorithm in [39]. A general overview of the PSRL algorithm can be found in Algorithm 3.

PSRL maintains a prior distribution on the set of MDPs $\mathcal{M}$. This contains the reward distribution (on $|\mathcal{S}||\mathcal{A}|$ variables) and the transition probability (on $|\mathcal{S}|^2|\mathcal{A}|$ variables).

Algorithm 3: Posterior Sampling for Reinforcement Learning (PSRL)

Input: prior distribution ϕ , episode length H

for *episode* $k = 1, 2, \ldots$ **do**
  sample MDP $M_k \sim \phi(\cdot \mid \mathcal{H}_{k1})$
  compute $\mu_k \in \operatorname{argmax}_{\mu} V_{\mu,1}^{M_k}$
  for *time* $h = 1, 2, \ldots, H$ **do**
    take action $a_{kh} = \mu_k(s_{kh}, h)$
    observe r_{kh} and s_{kh+1}
    update $\mathcal{H}_{kh} = \mathcal{H}_{kh} \cup (a_{kh}, r_{kh}, s_{kh+1})$
  end
end

At the beginning of each episode a MDP is sampled from the posterior distribution using the history. After that a set of optimal policies is found using backward induction, since at this point P^M and R^M are considered known we can use backward induction. The agent picks one of those optimal policies and executes that policy for the duration of that episode. Afterwards the witnessed rewards and transitions are used to update the history and this process is repeated for the next episode.

2.6 | Brief summary

We have introduced RL and gave an overview of 2 models: a bandit problem and a MDP. For the bandit problem we described three types of algorithms to find a policy: ϵ-greedy, UCB, and TS.

For RL on MDPs we have shown two types of algorithms: OFU–RL algorithms and PSRL. The main difference between these two algorithms is that the OFU based algorithms use a confidence set generator and then find the best possible model in this confidence set and find the policy on this model. The difficulty lies in ensuring that the confidence set contains the true MDP while also shrinking the confidence set over time so that as time progresses you converge to the true model. In PSRL we use the prior distribution to generate a single MDP and solve for this single MDP. If this prior distribution is an estimator for the true MDP, when time progresses we converge to the true MDP and therefore the optimal policy.

3

Clustering on Block Markov Chains

This Chapter investigates the inspiration behind this thesis; a paper called *Clustering in block Markov chains* [17]. The objective in [17] is to cluster states by using a single realisation of a Block Markov Chain walk. In a Block Markov Chain the clusters are determined by the transition probabilities of the states. Clustering is the assignment of every state to a cluster. First they introduce the Block Markov Chain framework and describe general properties that must be met for clustering to be possible in Block Markov Chains. Finally they introduce an algorithm that is able to cluster the states using a single trajectory. In this chapter we will briefly highlight the key foundations of the paper [17] for this thesis and report further research on the clustering algorithm.

In Section 3.1 the motivation and theory behind clustering on a Block Markov Chain trajectory will be described. This section concludes with two algorithms that can cluster states using just a single trajectory. Next, in Section 3.2, the Block Markov Chain simulator created for this thesis project will be described. This Chapter concludes with several Sections containing new insights into clustering in Block Markov Chains. In Section 3.3 a numerical analysis of the number of improvement steps is performed. In Section 3.4 the performance of the clustering algorithm is investigated on the set of possible BMC parameters. In Section 3.5 the difference in stochastic properties of two different stochastic processes is analysed; a process whose dynamics are generated by a pure Block Markov transition matrix and a process spawned by mixed Block Markov transition matrices. Finally, Section 3.6 compares the performance of the clustering algorithm on the mixed Block Markov chain to the performance on the pure Block Markov chain.

3.1 | Clustering on a Block Markov Chain trajectory

We now describe the work of [17]. We do this by first introducing the problem, the model definition of a Block Markov Chain (BMC), and general clustering results. After we introduce the key concepts the two algorithms found in [17] are explained in more detail. In this Chapter we use a different notation for various parameters compared to the original paper. We do this so that the notation used in this chapter does not interfere with the notation of the MDP theory. Our aim is to combine these two methodologies; see Chapter 4.

3.1.1 | Background

The goal of the paper [17] is to determine whether we can obtain the hidden cluster structure from a single observed state trajectory of length $T+1$. Such a trajectory or walk is illustrated in Figure 3.1. At every time instance the current state is denoted by S_t and the next state S_{t+1} is determined by transition matrix. The first objective of the paper is to obtain requirements on the problem parameters and minimum length T of the trajectory that makes clustering possible. The second objective was to create a clustering algorithm that reach the fundamental detectability limits. In [17] it is shown that when the number of possible states n tends to infinity the trajectory length has to be at least $n \log n$ for the clustering to be exact. For accurate clustering the trajectory has to be of length n. The definitions of accurate and exact will follow later on in this section.

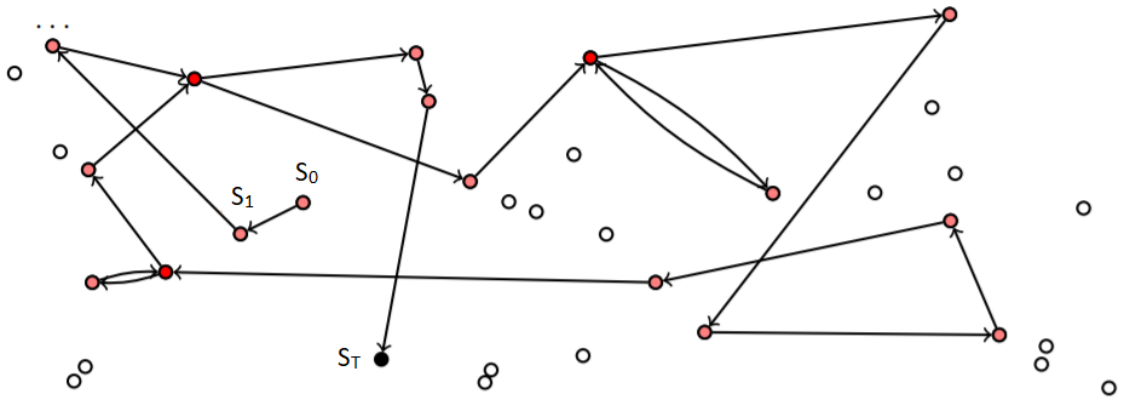


Figure 3.1: The objective of [17] is to infer the hidden cluster structure of the Markov chain from a single sample path $S_0, S_1, \dots, S_T$. Image courtesy of [40].

The BMC Obtaining the cluster structure turned out to be possible because the trajectory is generated by a Markov chain with a specific structure; specifically a BMC. A BMC is a Markov chain characterized by a block structure in its transition matrix. These blocks determine the clusters that have to be found by an algorithm. All states in the same block (cluster) have similar transition probabilities.

We now define the block Markov transition matrix and the dynamics of the BMC, which together allows us to generate a sample path as shown in Figure 3.1. We assume we have n states and denote the state space by $\mathcal{S} = \{1, \dots, n\}$. We assume there are K clusters and denote the cluster space by $\mathcal{C} = \{1, \dots, K\}$. A cluster is a collection of states and no state can be in more than one cluster. By $\sigma(i) \in \mathcal{C}$ we denote the cluster of the state i. The size of cluster k, i.e. the number of states in that particular cluster, is $|\mathcal{C}_k|$ for all $k \in \mathcal{C}$. We define p_{k_1,k_2} for $k_1, k_2 \in \mathcal{C}$ as the intercluster transition probability between clusters k_1 and k_2.

We now define the probability of going to a next state S_{t+1}, given the current state S_t for all states in the set $\mathcal{S}$. We capture these dynamics in the transition matrix P. All entries of this matrix are determined by the probability to go from a state x to state y by [17]

$$P_{x,y} \triangleq \mathbb{P}(S_{t+1} = y \mid S_t = x) = \frac{p_{\sigma(x),\sigma(y)}}{|\mathcal{C}_{\sigma(y)}| - \mathbb{1}(\sigma(x) = \sigma(y))} \mathbb{1}(x \neq y) \quad \text{for all} \quad x, y \in \mathcal{S}\,. \tag{3.1}$$

Here $\mathbb{1}$ denotes the indicator function. To illustrate how the transition matrix looks like, a generic transition matrix for K clusters is displayed in (3.2). Here all rows and columns are ordered so that all the states belonging to the same cluster are displayed next to one another. The width of the first block is $|\mathcal{C}_1|$ elements, the width of the second block is $|\mathcal{C}_2|$ elements and so on up to the K-th block with a width of $|\mathcal{C}_K|$ elements. Note that the transition matrix has several blocks in it, hence the name BMC.

$$P = \left(\begin{array}{cccc|cccc|c|cccc}
0 & \frac{p_{1,1}}{|\mathcal{C}_1|-1} & \cdots & \frac{p_{1,1}}{|\mathcal{C}_1|-1} & \frac{p_{1,2}}{|\mathcal{C}_2|} & \frac{p_{1,2}}{|\mathcal{C}_2|} & \cdots & \frac{p_{1,2}}{|\mathcal{C}_2|} & \cdots & \frac{p_{1,K}}{|\mathcal{C}_K|} & \frac{p_{1,K}}{|\mathcal{C}_K|} & \cdots & \frac{p_{1,K}}{|\mathcal{C}_K|} \\
\frac{p_{1,1}}{|\mathcal{C}_1|-1} & 0 & \cdots & \frac{p_{1,1}}{|\mathcal{C}_1|-1} & \frac{p_{1,2}}{|\mathcal{C}_2|} & \frac{p_{1,2}}{|\mathcal{C}_2|} & \cdots & \frac{p_{1,2}}{|\mathcal{C}_2|} & \cdots & \frac{p_{1,K}}{|\mathcal{C}_K|} & \frac{p_{1,K}}{|\mathcal{C}_K|} & \cdots & \frac{p_{1,K}}{|\mathcal{C}_K|} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\
\frac{p_{1,1}}{|\mathcal{C}_1|-1} & \frac{p_{1,1}}{|\mathcal{C}_1|-1} & \cdots & 0 & \frac{p_{1,2}}{|\mathcal{C}_2|} & \frac{p_{1,2}}{|\mathcal{C}_2|} & \cdots & \frac{p_{1,2}}{|\mathcal{C}_2|} & \cdots & \frac{p_{1,K}}{|\mathcal{C}_K|} & \frac{p_{1,K}}{|\mathcal{C}_K|} & \cdots & \frac{p_{1,K}}{|\mathcal{C}_K|} \\
\hline
\frac{p_{2,1}}{|\mathcal{C}_1|} & \frac{p_{2,1}}{|\mathcal{C}_1|} & \cdots & \frac{p_{2,1}}{|\mathcal{C}_1|} & 0 & \frac{p_{2,2}}{|\mathcal{C}_2|-1} & \cdots & \frac{p_{2,2}}{|\mathcal{C}_2|-1} & \cdots & \frac{p_{2,K}}{|\mathcal{C}_K|} & \frac{p_{2,K}}{|\mathcal{C}_K|} & \cdots & \frac{p_{2,K}}{|\mathcal{C}_K|} \\
\frac{p_{2,1}}{|\mathcal{C}_1|} & \frac{p_{2,1}}{|\mathcal{C}_1|} & \cdots & \frac{p_{2,1}}{|\mathcal{C}_1|} & \frac{p_{2,2}}{|\mathcal{C}_2|-1} & 0 & \cdots & \frac{p_{2,2}}{|\mathcal{C}_2|-1} & \cdots & \frac{p_{2,K}}{|\mathcal{C}_K|} & \frac{p_{2,K}}{|\mathcal{C}_K|} & \cdots & \frac{p_{2,K}}{|\mathcal{C}_K|} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\
\frac{p_{2,1}}{|\mathcal{C}_1|} & \frac{p_{2,1}}{|\mathcal{C}_1|} & \cdots & \frac{p_{2,1}}{|\mathcal{C}_1|} & \frac{p_{2,2}}{|\mathcal{C}_2|-1} & \frac{p_{2,2}}{|\mathcal{C}_2|-1} & \cdots & 0 & \cdots & \frac{p_{2,K}}{|\mathcal{C}_K|} & \frac{p_{2,K}}{|\mathcal{C}_K|} & \cdots & \frac{p_{2,K}}{|\mathcal{C}_K|} \\
\hline
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
\hline
\frac{p_{K,1}}{|\mathcal{C}_1|} & \frac{p_{K,1}}{|\mathcal{C}_1|} & \cdots & \frac{p_{K,1}}{|\mathcal{C}_1|} & \frac{p_{K,2}}{|\mathcal{C}_2|} & \frac{p_{K,2}}{|\mathcal{C}_2|} & \cdots & \frac{p_{K,2}}{|\mathcal{C}_2|} & \cdots & 0 & \frac{p_{K,K}}{|\mathcal{C}_K|-1} & \cdots & \frac{p_{K,K}}{|\mathcal{C}_K|-1} \\
\frac{p_{K,1}}{|\mathcal{C}_1|} & \frac{p_{K,1}}{|\mathcal{C}_1|} & \cdots & \frac{p_{K,1}}{|\mathcal{C}_1|} & \frac{p_{K,2}}{|\mathcal{C}_2|} & \frac{p_{K,2}}{|\mathcal{C}_2|} & \cdots & \frac{p_{K,2}}{|\mathcal{C}_2|} & \cdots & \frac{p_{K,K}}{|\mathcal{C}_K|-1} & 0 & \cdots & \frac{p_{K,K}}{|\mathcal{C}_K|-1} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\
\frac{p_{K,1}}{|\mathcal{C}_1|} & \frac{p_{K,1}}{|\mathcal{C}_1|} & \cdots & \frac{p_{K,1}}{|\mathcal{C}_1|} & \frac{p_{K,2}}{|\mathcal{C}_2|} & \frac{p_{K,2}}{|\mathcal{C}_2|} & \cdots & \frac{p_{K,2}}{|\mathcal{C}_2|} & \cdots & \frac{p_{K,K}}{|\mathcal{C}_K|-1} & \frac{p_{K,K}}{|\mathcal{C}_K|-1} & \cdots & 0
\end{array}\right) \tag{3.2}$$

To illustrate the cluster structure and to visualise the transition matrix in an example we show the situation for two clusters in Figure 3.2. Given that we are in the current state S_t, that is in cluster $\mathcal{C}_1$, we jump to cluster $\mathcal{C}_2$ with the intercluster probability $p_{1,2}$. By symmetry the probability of jumping to any particular state $s \in \mathcal{C}_2$ is scaled by the size of this cluster $|\mathcal{C}_2|$. The probability to jump to another state within the same cluster follows in an identical fashion, however, we do not allow self jumps and compensate for that, by dividing the intercluster probability $p_{1,1} = 1 - p_{1,2}$ by the size of the first cluster minus one, i.e., $|\mathcal{C}_1| - 1$. This Figure indicates how the dynamics of a BMC work and that the clusters we want to detect are a collection of states.

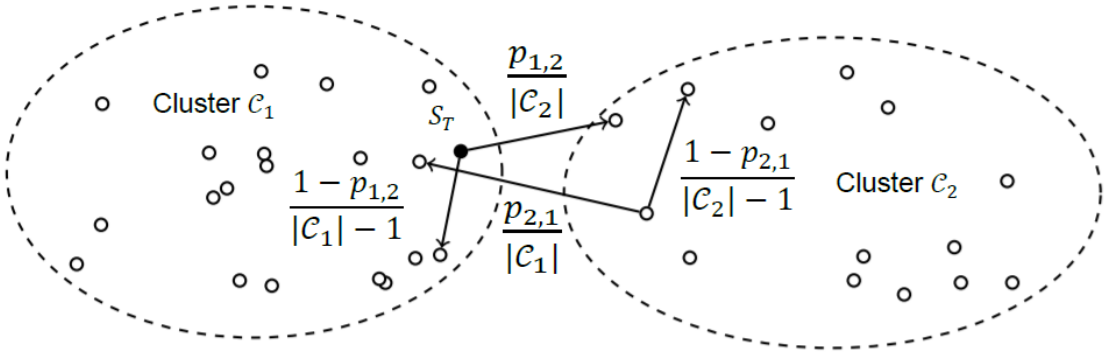


Figure 3.2: Illustration of a BMC with $K = 2$ clusters. When the Markov chain is at a state $S_t \in \mathcal{C}_1$, the next state will be in $\mathcal{C}_2$ with probability $p_{1,2}$ and it will jump to another state in $\mathcal{C}_1$ with probability $1 - p_{1,2}$. Figure adjusted from [17].

Fundamental detectability limit Recall that the objective of [17] is to accurately detect the clusters and to determine what the required trajectory length is to achieve accurate detection. Suppose the output of some clustering algorithm are estimated cluster assignments $\hat{\mathcal{C}}_i$ for $i \in 1, \ldots, K$. We define $\mathcal{E}$ as the set of misclassified states, by

$$\mathcal{E} \triangleq \bigcup_{k=1}^{K} \hat{\mathcal{C}}_{\gamma^{opt}(k)} \backslash \mathcal{C}_k \quad \text{where} \quad \gamma^{opt} \in \arg\min_{\gamma \in \text{Perm}(K)} \left| \bigcup_{k=1}^{K} \hat{\mathcal{C}}_{\gamma(k)} \backslash \mathcal{C}_k \right|. \tag{3.3}$$

We now use this set of misclassified states to define two measures of detection performance of any clustering algorithm. If we talk about *exact detection* of the clusters (exact recovery) we imply that the number of misclassified states $|\mathcal{E}|$ tends to zero. If we talk about *accurate detection* we imply that the fraction of misclassified states $|\mathcal{E}|/n$ tends to zero.

In [40] the analysis is done for the number of states tending to infinity, $n \to \infty$, this is called asymptotically. They find conditions that have to be met so that there may exist a clustering algorithm that achieves asymptotically accurate ($|\mathcal{E}|/n \to 0$) or exact ($|\mathcal{E}| \to 0$) detection.

The approach of [17] is to bound the number of misclassified states and the statements in [17] do just that. The main theorem that lower bounds the size of $\mathcal{E}$ holds for *any* clustering algorithm. We reproduce [17, Theorem 1] here:

Theorem 3 (Sanders, Proutière, Yun, 2018)**.** *Assume that $T = \omega(n)$. Then there exists a strictly positive and finite constant C independent of n such that: for any clustering algorithm*

$$\mathbb{E}\left[|\mathcal{E}|\right] \geq Cn \exp\left(-I(\alpha, p)\frac{T}{n}(1 + o(1))\right).$$

In Theorem 3 the information quantity $I(\alpha, p)$ is defined as

$$I(\alpha, p) \triangleq \min_{a \neq b}\Big\{\sum_{k=1}^{K} \frac{1}{\alpha_a}\big(\pi_a p_{a,k} \ln \frac{p_{a,k}}{p_{b,k}} + \pi_k p_{k,a} \ln \frac{p_{k,a}\alpha_b}{p_{k,b}\alpha_a}\big) + \big(\frac{\pi_b}{\alpha_b} - \frac{\pi_a}{\alpha_a}\big)\Big\}. \tag{3.4}$$

Here α denotes the vector of fraction of states that belong to every cluster and the matrix p contains the intercluster transition probabilities. Furthermore the vector π contains the equilibrium probability of a cluster. In particular π is the solution to the equation $\pi^T p = \pi^T$.

Theorem 3 depends on an information quantity $I(\alpha, p) \geq 0$ that measures how difficult it is to cluster in a BMC. The exact mathematical conditions of asymptotically accurately, $\mathbb{E}\left[|\mathcal{E}|\right] = o(n)$, and asymptotically exact, $\mathbb{E}\left[|\mathcal{E}|\right] = o(1)$, are derived from this bound. The bound also provides the necessary conditions for a clustering algorithm to be asymptotically exact or accurate. The necessary conditions for the existence of an algorithm that has asymptotically accurate detection are $I(\alpha, p) > 0$ and $T = \omega(n)$. The necessary conditions for the existence of an asymptotically exact algorithm are $I(\alpha, p) > 0$ and $n \log n / I(\alpha, p) = \omega(1)$. This implies that T has to be at least the order of $n \log n$. Based on these definitions we define three regimes for the trajectory length. The *sparse regime* is defined as $n < T < n \log n$, here there may exist an algorithm with asymptotically accurate detection. The *critical regime* is defined as T in the order of $n \log n$. Here. exact detection is just possible. Last we define the *dense regime* where $T = \omega(n \log n)$. To recap the necessary conditions with this regimes: if we are in the sparse regime, $T = \omega(n)$ and $I(\alpha, p) > 0$, then asymptotically accurate detection is possible, if we are in the dense regime, $T = \omega(n \log n)$, and $I(\alpha, p) > 0$, then asymptotically exact detection is possible.

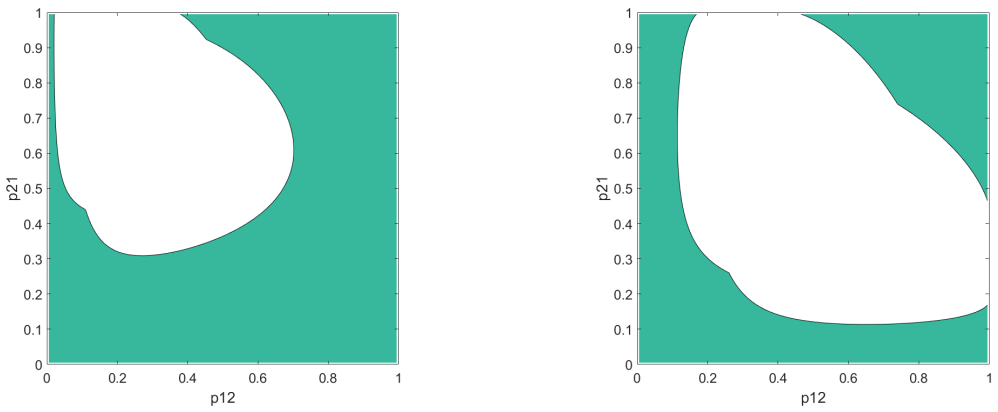
Critical regime We are now going to look at what the conditions on $I(\alpha, p)$ are for asymptotically exact detection in the critical regime. We set the trajectory length $T = n \log n$. For asymptotically exact detection we require the right-hand side of Theorem 3 to satisfy

$$\lim_{n \to \infty} C \exp((1 - (1 + o(1))I(\alpha, p)) \log n) = 0\,. \tag{3.5}$$

This leads to a condition on $I(\alpha, p)$ in order for some clustering algorithm, able to do exact recovery, to exist. Specifically, (3.5) tends to zero if $I(\alpha, p) > 1$. We can thus find a region for which asymptotic exact recovery in the critical regime is possible by analysing the function $I(\alpha, p)$. We will call this region the feasibility region $\mathcal{F}$ and define it as

$$\mathcal{F} \triangleq \{(\alpha, p) \mid I(\alpha, p) > 1\}. \tag{3.6}$$

To illustrate the feasibility region we study the case of two clusters. We study the set of parameters (α, p) for which $I(\alpha, p) \geq 1$. In Figure 3.3 the feasibility region is plotted for the case of $K = 2$ clusters with $\alpha_2 = 0.25$ (left) and $\alpha_2 = 0.5$ (right). Recall that α_2 is the fraction of states in cluster two. The plot in Figure 3.3 is made for all parameters $(p_{12}, p_{21}) \in (0, 1)^2$. The feasibility region is shown in green.



(a) $\mathcal{F}$ for $\alpha_2 = 0.25, p \in (0,1)^2$ (b) $\mathcal{F}$ for $\alpha_2 = 0.5, p \in (0,1)^2$

Figure 3.3: Feasibility region $\mathcal{F}$ where asymptotically exact recovery by some clustering algorithm may be possible, given that there are $K = 2$ clusters. The relative cluster sizes 1 and 2 are $\alpha_2 = 0.25$, $\alpha_1 = 0.75 = 1-\alpha_2$ (left) and on the right $\alpha_2 = \alpha_1 = 0.5$.

We have now described the foundations of clustering in Block Markov Chains, and we briefly explained the reasoning behind it and what the conditions are for a clustering algorithm to potentially exist. In [40] an algorithm is also developed that meets this lower bound on the number of misclassified states up to a constant. We will now briefly explain this algorithm.

3.1.2 | Algorithms

The approach used by [40] is an algorithm that consists of two steps. The two steps can be formulated as two individual algorithms: the Spectral Clustering Algorithm (SCA) and the Cluster Improvement Algorithm (CIA). The full two-step approach from [40] will be referred to as the CA. The first step, the CIA, is ran only once. The second step, the CIA, is an update step and can be ran multiple times to likely converge to a better cluster assignment. The CA has as input the number of clusters K, the number of states n, and the trajectory $S_0, \dots, S_{T-1}$. Based on this input all states $\{1, \dots, n\}$ will be placed into one of the K clusters. All states assigned to cluster 1 will be denoted by $\hat{\mathcal{C}}_1$, all state assigned to cluster 2 by $\hat{\mathcal{C}}_2$, and so on.

The pseudo-code for the SCA can be found in Algorithm 4. The aim of the SCA is to provide a good first estimate on the cluster assignment. The SCA computes the cluster assignment by computing the number of state transitions $\hat{N}_{x,y} \triangleq \sum_{t=0}^{T-1} \mathbb{1}[s_t = x, s_{t+1} = y]$ from all states $x \in \mathcal{S}$ to each state $y \in \mathcal{S}$. From the $\hat{N}$ matrix a rank-K approximation is made by using the basis vectors of the Singular Value Decomposition (SVD). Using this rank-K approximation a K-means clustering algorithm is applied. This K-means algorithm yields the cluster assignment.

Algorithm 4: Pseudo-code for Spectral Clustering Algorithm

Input : A trajectory $S_0, S_1, \dots, S_T$, the number of states n, the number of clusters K
Output: A cluster assignment $\hat{\mathcal{C}}_1, \dots, \hat{\mathcal{C}}_K$ and matrix $\hat{N}$

for $x \leftarrow 1$ *to* n **do**
  for $y \leftarrow 1$ *to* n **do**
    $\hat{N}_{x,y} \leftarrow \sum_{t=0}^{T-1} \mathbb{1}[S_t = x, S_{t+1} = y]$
  end
end
Calculate the singular value decomposition, $U\Sigma V^T$, of $\hat{N}$
Order U, Σ, V based on singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$
Construct a rank-K approximation $\hat{R} = \sum_{k=1}^{K} \sigma_k U_{\cdot,k} V_{\cdot,k}^T$
Apply a K-means algorithm to $\hat{R}$ to determine cluster assignment $\hat{\mathcal{C}}_1, \dots, \hat{\mathcal{C}}_K$

The SCA alone is capable of achieving asymptotically accurate detection whenever this is possible

at all. This can be seen from [17, Theorem 2] which we reproduce here:

Theorem 4 (Sanders, Proutière, Yun, 2018)**.** *Assume that $T = \omega(n)$ and $I(\alpha, p) > 0$. Then the proportion of misclassified states after the Spectral Clustering Algorithm satisfies:*

$$\frac{|\mathcal{E}|}{n} = O\left(\frac{n}{T}\ln\frac{T}{n}\right) = o(1)\,.$$

However, this theorem fails at ensuring exact detection, even in the dense regime. To make exact detection possible the second step of the clustering algorithm is needed.

The second step improves a current cluster assignment. The pseudo-code of the CIA can be found in Algorithm 5. The improvement of the cluster assignment is done by maximizing a likelihood ratio and placing each state in the cluster that best describes the sample path given our current belief of the BMC parameters. The log likelihood ratio is a combination of estimated parameters from the BMC, i.e., the estimated intercluster probabilities and the estimated probability of a state to be in a cluster. The likelihood function can be found in line 9 of Algorithm 5. In the CIA we use the estimated a sum of (a part) of the estimated transition matrix $\hat{N}$, we define $\hat{N}_{A,B} = \sum_{x\in A}\sum_{y\in B}\hat{N}_{x,y}$. Here A and B are subsets of $\mathcal{C} = \{1, \dots, n\}$ that spawns the full dimension of $\hat{N}$.

Algorithm 5: Pseudo-code for Cluster Improvement Algorithm

Input : A cluster assignment $\hat{\mathcal{C}}_1^{[h]}, \dots, \hat{\mathcal{C}}_K^{[h]}$, and $\hat{N}$
Output: An updated cluster assignment $\hat{\mathcal{C}}_1^{[h+1]}, \dots, \hat{\mathcal{C}}_K^{[h+1]}$

1 $n \leftarrow \dim(\hat{N})$, $\mathcal{C} \leftarrow \{1, \dots, n\}$, $T \leftarrow \sum_{x,y}\hat{N}_{x,y}$
2 **for** *a ←1 to K* **do**
3 $\quad\hat{\pi}_a \leftarrow \hat{N}_{\hat{\mathcal{C}}_a^{[h]},\mathcal{C}}/T$, $\hat{a}_a \leftarrow |\hat{\mathcal{C}}_a^{[h]}|/n$, $\hat{\mathcal{C}}_a^{[h+1]} \leftarrow \emptyset$
4 $\quad$**for** *b ←1 to K* **do**
5 $\quad\quad\hat{p}_{a,b} \leftarrow \hat{N}_{\hat{\mathcal{C}}_a^{[h]},\hat{\mathcal{C}}_b^{[h]}}/\hat{N}_{\hat{\mathcal{C}}_a^{[h]},\mathcal{C}}$
6 $\quad$**end**
7 **end**
8 **for** *s ←1 to n* **do**
9 $\quad c_s^{opt} \leftarrow \operatorname{argmax}_{c=1,\dots,K}\{\sum_{k=1}^{K}(\hat{N}_{s,\hat{\mathcal{C}}_k^{[h]}}\ln\hat{p}_{c,k} + \hat{N}_{\hat{\mathcal{C}}_k^{[h]}}\ln\frac{\hat{p}_{k,c}}{\hat{a}_c}) - \frac{T}{n}\frac{\hat{\pi}_c}{\hat{a}_c}\}$
10 $\quad\hat{\mathcal{C}}_{c_s^{opt}}^{[h+1]} \leftarrow \hat{\mathcal{C}}_{c_s^{opt}}^{[h+1]} \cup \{s\}$
11 **end**

The CIA is proven to let the number of misclassified states tend to zero when the number of times the CIA is ran h, together with the number of states n, tend to ∞. For this to be possible the initial cluster assignment must be close enough to the true underlying cluster assignment. It turns out that the SCA algorithm gives a sufficiently accurate initial cluster assignment. When the CIA is initiated with the SCA, the CA allows for asymptotically exact recovery in the dense regime. This is shown by [17, Theorem 3] which we reproduce here:

Theorem 5 (Sanders, Proutière, Yun, 2018)**.** *Assume that $T = \omega(n)$ and $I(\alpha, p) > 0$. Then there exists a constant $C > 0$ such that for any $h \geq 1$, after h iterations of the Clustering Improvement Algorithm, initially applied to the output of the Spectral Clustering Algorithm, we have:*

$$\frac{|\mathcal{E}^{[h]}|}{n} = O\left(e^{-h(\ln\frac{T}{n} - \ln\ln\frac{T}{n})} + e^{-C\frac{T}{n}I(\alpha,p)}\right).$$

If the number of times the CIA step is ran is $h = \ln n$, the number of misclassified states tends to the fundamental recovery limit of Theorem 3 up to the constant C. The number of improvement steps in the range of $\ln n$ hence appears to be a good guideline for the number of times the improvement step has to be run in a general setting. All in all, the combination of these two algorithms ensures an algorithm that is able to asymptotically (as $n \to \infty$) recover the clusters exactly.

3.2 | BMC simulator

One of the thesis project goals is to implement a Block Markov Chain simulator that includes the two-stage spectral clustering algorithm of [40]. This simulator is build to evaluate and do extensive testing on the CA. At first we implemented our BMC simulator's basic functions in both Matlab and Python. But as this project progressed solely the Python implementation was developed further. The Python implementation was inspired from the tabulaRL environment[1]. The simulator is build upon two classes: an environment class and an agent class. Additionally we created various functions to analyse the performance of the various experiments.

3.2.1 | Environment Class

The environment class contains the dynamics of the Block Markov Chain and is the basis of the trajectory generation. By default the class only stores the current state and the current time step. Depending on the options provided when initialising the class the total state trajectory up to that time step is stored. This class has 3 callable functions; the initialise function, the advance function and the reset function.

- The initialise function creates the environment. The input of this function is the transition matrix. Furthermore there are extra arguments to store the total trajectory and to set the initial state of the walk. After storing the transition matrix and setting the current time step to zero, this function will set the current state to either the input state, or to a uniformly at random selected state.
- The advance function lets the walk advance one step. It moves the current state to the next state, and the current time step is raised by one.
- The reset function maintains the transition dynamics but resets the time step to zero and the current state to the initially started at state or an uniform random selected state.

Besides to the the environment class there are some supporting functions to help generate the input for this class. For example, we have a function that generates transition matrices based on the number of states per cluster and the intercluster transition probabilities.

3.2.2 | Agent Class

The Agent class contains the agent for the BMC simulator. The agent stores the state trajectory and contains the implementation of the SCA and the CIA. The agent class has two callable functions: the initialise function and the update function.

- The initialise function creates the agent. The input of this function are the trajectory length, the number of clusters, the number of states, and the number of times CIA has to be run.
- The update function. The input of the update function are the new state that has to be added to the trajectory and the current time step. If the current time step is equal to the clustering length the agent will run the clustering algorithm on the stored trajectory. It runs the clustering algorithm with the parameters given to the agent by initialisation. The cluster assignment per state will be stored by the agent.

3.2.3 | Utility functions

Besides these two classes, some utility functions are implemented to test and validate the performance. For example, these include a function to generate the feasibility region and a function to calculate the set of misclassified states based on input of the environment and the cluster assignment of the agent. This category also includes the functions that will show up in the following sections where experimental results are discussed.

3.3 | Number of improvement steps

The fundamental recovery limit in Theorem 3 in combination with the performance of the CIA in Theorem 5 suggest that the number of times the improvement step has to be run on the order of $\sim \ln n$. In the numerical results in [40] it is not immediately clear that this is actually necessary. In all simulations there is no clear sign of improvement after the third improvement step. We will investigate the necessary number of improvement steps in this section. This is a question that was raised by referees during the review process of [40]. Besides investigating this scientific question, if we can conclude that a smaller number of improvement steps is required for identical performance, the computational cost

[1] https://github.com/iosband/TabulaRL

of the CA is reduced. The improvement step, the CIA, is namely the computational heaviest part of the CA.

In this section we will show that indeed a number of improvement steps h in the order of $h \sim \ln n$ is required number of improvement steps for exact recovery if we are in the critical regime. We investigate general convergence but also investigate the convergence to an accurate assignment, when the accuracy $\nu > 0.95$. When the trajectory length increases towards the dense regime, a smaller number of iterations of the CIA are needed to converge towards a cluster assignment for the general trials and the trials that resulted in an accurate assignment.

We test the influence of the number of improvement steps by looking at the estimated cluster assignment after every improvement step. We compare this output with the previous three estimated cluster assignments. If we observe an identical assignment we define that as convergence of the algorithm and set the number of improvement steps for the algorithm to converge to the first time the cluster assignment was found. We look at the previous three estimates because empirically we observed that a cluster assignment sometimes convergence in the sense that it alternates between two different assignments or in a minor amount of cases a rotation of three different assignments. By only looking at the previous three assignments we neglect all cycles larger than three but we assume that this portion is neglectable.

For this experiment we use the same transition matrix and cluster sizes as in the numerical experiment of section 4.1 of the original paper by [40]. We have $n = 300$ states grouped into three clusters of sizes $|\mathcal{C}_1| = 48$, $|\mathcal{C}_2| = 93$ and $|\mathcal{C}_3| = 159$. The intercluster transition probabilities are defined by $p = (0.9200, 0.0450, 0.0350; 0.0125, 0.8975, 0.0900; 0.0175, 0.0200, 0.9625)$. For this number of states the expected number of improvement steps is $\ln n = \ln 300 \approx 5.7$. First we will investigate the critical trajectory length of $T = n \ln n \approx 1711$. We run the clustering algorithm on a thousand different trajectories (trials) generated by this BMC with a maximum of ten improvement steps. The histogram of the number of improvement steps where the cluster assignment converges is shown in Figure 3.4 in blue. Observe that for a relevant portion of the trials the algorithm does not converge to a cluster assignment, indicated by the histogram at the number of improvement steps of fifteen. On top of that, we also have to remark that convergence does not necessarily mean that the found cluster assignment is exact or accurate. By looking at the observed data we empirically noticed that if the assignment from the first spectral step, the SCA, is not accurate enough the algorithm sometimes converges to a cluster assignment that is not accurate, i.e. an accuracy ($\nu < 0,95$).

If the trial resulted in an accurate cluster assignment, i.e., the trials that after ten improvement steps have an accuracy higher than $\nu = 0.95$, or equivalently more than 285 states clustered correctly, we highlighted these trials in red in the histogram. We can observe that all of these accurate trials converge within nine improvement steps. However, in a relevant portion of the red trials the algorithm converges before the estimated number of improvement steps.

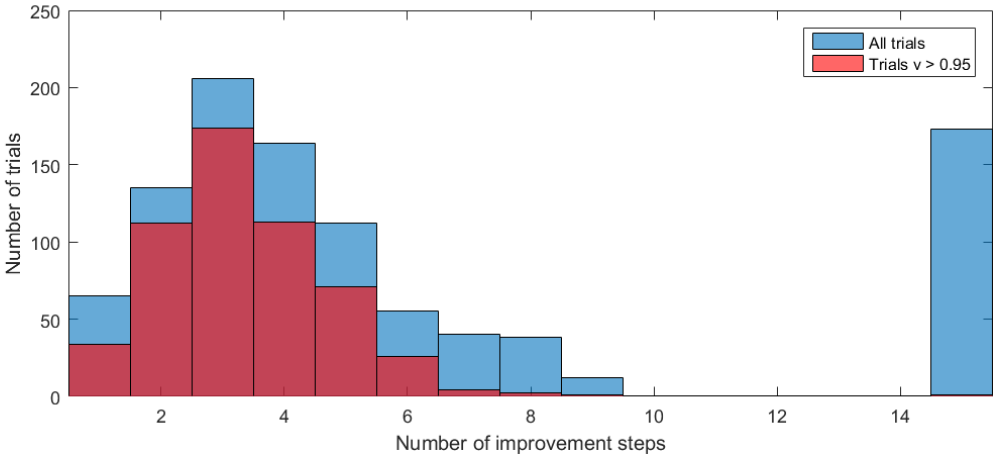


Figure 3.4: The number of improvement steps that the clustering algorithms needs before the cluster assignment is no longer changed. We have $n = 300$ states and the number of expected improvement steps is $\ln n \approx 5.7$. The maximum number of improvement steps is 10. For the trials with the bar at 15 no convergence in the cluster assignment is found within 10 improvement steps.

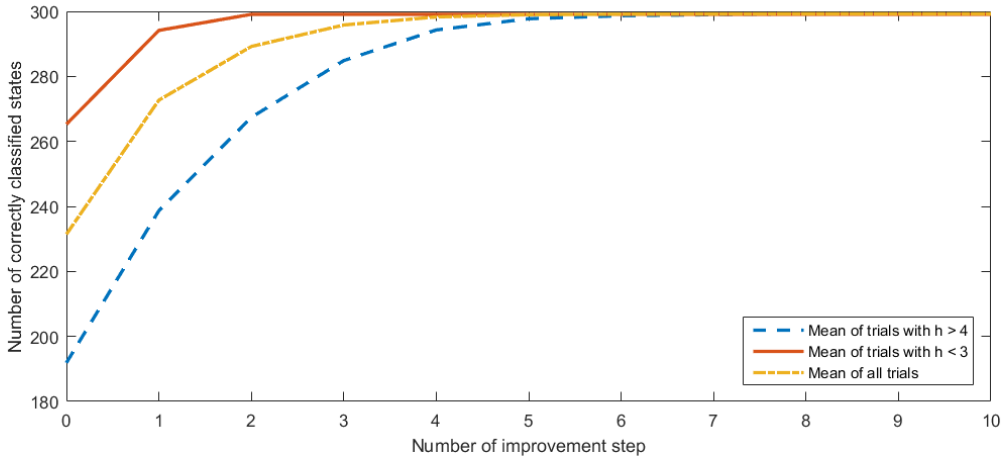


Figure 3.5: Averaged number of correctly classified clusters of the trials that converge to a good cluster assignment. The average is shown for all trials, trials that converge in less than three improvement steps, and trials that converge in more than four improvement steps.

We are now going to investigate how the accurate cluster assignments converge. We just take the trials that converge to an accurate assignment and plot the average number of correctly classified states after every improvement step. The dash-dot yellow line in Figure 3.5 demonstrates the average correctness of the cluster assignment. Next to the average over all trials we also plotted the average behaviour when a low number of improvement steps (h<3) and a larger number of improvement steps are needed (h>4). From the difference between those two average curves we conclude that the initial cluster assignment is critical in the number of improvement steps that are needed to converge to the final cluster assignment.

We remark that in a real application we would not have the real cluster assignment and therefore the accuracy would not be available to the agent. However, based on these results, if the clustering algorithm does not show convergence in the cluster assignment after $\ln n$ improvement steps, we could, from a practical point of view, wait for a longer period of time before clustering or neglect this observation in total because we can suspect that the assignment is not accurate.

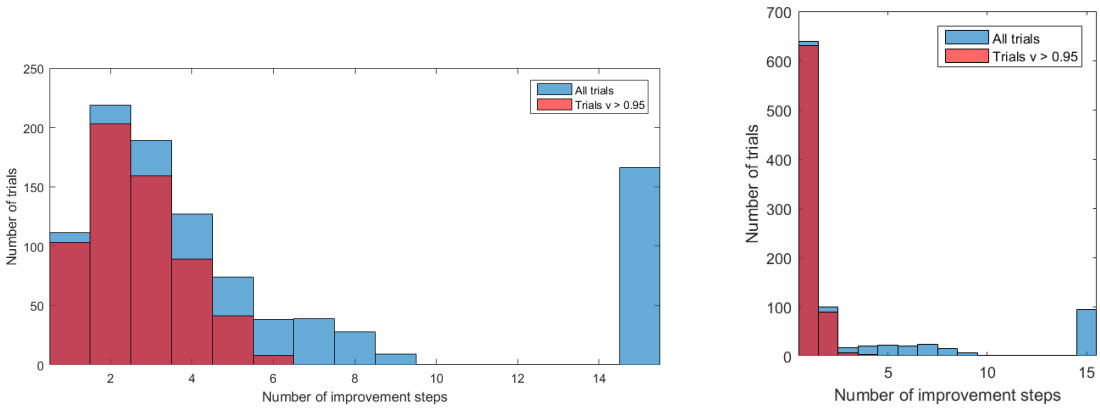
Now that we have looked at the convergence of the cluster assignment in the critical regime, we will next take a look at this convergence for larger trajectory lengths. We take the clustering length taken in the original paper, $T = 1973$ and we will also take a clear example of the dense regime, namely $T = 3000$. For both situations the histograms can be found in Figure 3.6. If we round the estimated number of improvement steps $\ln n \approx 5.7 \approx 6$. We can see from Figure 3.6a that for the trials with a high accuracy all of these trials converged within this estimate. If we take the dense regime from Figure 3.6b the maximum number of steps is four in the accurate case, but the majority converges within two improvement steps.

When we compare Figure 3.4 with Figure 3.6 we deduce that the increase in clustering length makes the spectral step more accurate. This is expected based on Theorem 4. Given that we have a better initial spectral estimate, the number of improvement steps required for convergence becomes lower.

Based on the trajectories that converge to an accurate cluster assignment, the red histograms in Figure 3.4 and Figure 3.6, we can conclude that the estimated number of improvement steps in the range of $\sim \ln n$ is an accurate number when we are in the critical regime. When we increase the trajectory length into range of the dense regime it appears that a smaller number of improvement steps is needed for convergence. We remark that while this study is conducted on a single BMC, it is sufficient to illustrate the statement that there exist a BMC for which in the critical regime the CA requires $\ln n$ number of improvement steps to converge.

3.4 | Performance Analysis of the CA in the critical regime

The goal of this section is to analyse the performance of the CA [40]. This analysis will be done in the critical regime for the case of $K = 2$ clusters. Analysing the critical regime gives insight into the



(a) $T = 1973$. (b) $T = 3000$.

Figure 3.6: Histograms of the number of improvement steps required for convergence of the cluster assignment for 1000 trials. The histograms are shown for $T = 1973$ (left) and $T = 3000$ (right).

behaviour of the CA when it is most difficult to have exact recovery. Additionally this gives insight into the phase transition ($I(\alpha, p) < 1, I(\alpha, p) > 1$) that occurs. Consider also that in a combined clustering MDP approach you want to learn the clusters as fast as possible, so T should be as short as possible to effectively reduce the exploration stage, and leave room for a longer exploitation stage. Thus the objective of this section is to answer the question:

> What is the performance of the Clustering Algorithm in the critical regime, and is this performance adequate for implementation in a BMC–MDP approach?

In Section 3.4.1 the metrics used to analyse the performance of the CA will be defined. In Section 3.4.2 the setup and results of various numerical simulations of the Block Markov Chain Clustering algorithm are shown and discussed. Finally in Section 3.4.3 conclusions are drawn for the general performance and behaviour of the CA.

3.4.1 | Metrics

To measure the performance of the clustering algorithm in the critical regime we are going to numerically estimate the feasibility region in Figure 3.3. We define the *accuracy* ν, given by the fraction of correctly classified states

$$\nu(\alpha, p) \triangleq 1 - \frac{|\mathcal{E}|}{n}. \tag{3.7}$$

Here $|\mathcal{E}|$ denotes the number of misclassified states and n is the total number of states. The accuracy ν can then be used to estimate a feasibility region $\hat{\mathcal{F}}$ of an algorithm given by

$$\hat{\mathcal{F}}(\epsilon) \triangleq \{(\alpha, p) \mid \mathbb{E}[\nu] \geq 1 - \epsilon\}, \tag{3.8}$$

which we will use as a proxy for a true estimator of $\mathcal{F}$: $\hat{\mathcal{F}}$ does not directly estimate $\mathcal{F}$. Intuitively, we expect recovery of the clusters to be possible if the accuracy is one and thus when ϵ goes to zero. This would make the algorithm have asymptotically accurate detection as the accuracy describes the fraction of correctly classified states and not the exact number of classified states. The feasibility region relies on asymptotic behaviour of the number of states, i.e. when $n \to \infty$. To compensate for a finite number of states we use the error ϵ to compensate for this asymptotic behaviour in the estimator (3.8).

3.4.2 | Setup, results, and discussion

All simulations in this section are done for the case that there are $K = 2$ clusters. In the case of two clusters the entire BMC can be described using the parameters n, α_2, p_{12} and p_{21}. Here n denotes the number of states and α_2 is the fraction of the total number of states that is in cluster two. The fraction of states in cluster one follows from this and equals $1 - \alpha_2$. The inter-cluster transition probabilities are given by p_{12} and p_{21}, and thus because p must constitute a stochastic matrix $p_{11} = 1 - p_{12}$ and

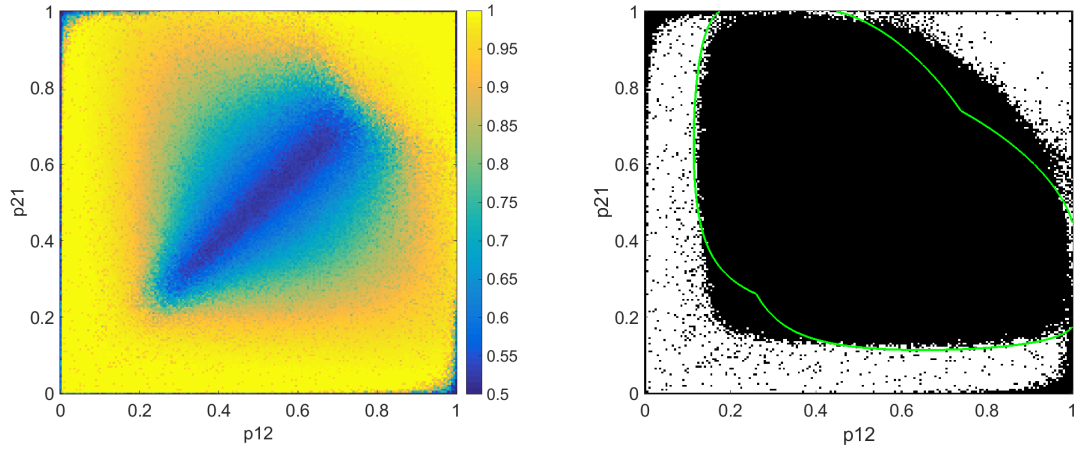
$p_{22} = 1 - p_{21}$. In this first part of the simulations n is set to 300 states. This results in a trajectory length of $T = n \log n \approx 1711$.

In this evaluation the parameter space $(p_{12}, p_{21}) \in (0, 1)^2$ is being rasterized. The measures that are used to visualise the performance for the CA is the accuracy ν in (3.7), and the estimated feasibility region $\hat{\mathcal{F}}$ from (3.8). The cluster improvement step of the CA is run six times in this implementation of the CA. The number of six improvement steps is based on the asymptotic result in the dense regime which allows exact recovery if the number of improvement steps is in the range of $\sim \log n$ [40, theorem 3], ans substantiated by our results in Section 3.3.

In Figure 3.7a the sample mean accuracy ν is shown calculated from ten trials for every pair (p_{12}, p_{21}) in the raster. The raster consists of a 2D-grid with step size 0.005. The lower bound on the accuracy in the case of two clusters is 0.5. To see this consider (3.3): if the number of correctly classified states would be lower than 0.5, say k_1, a permutation could be done in the labelling of the cluster assignment. After such relabelling the accuracy would be $1 - k_1$, and consequently higher than 0.5. On the right the estimated feasibility region can be seen for $\epsilon = 0.027$, the green line indicates the region where $I(\alpha, p) = 1$.

From Figure 3.3b and Figure 3.7b it can be noted that the feasibility regions are symmetric along the diagonal $p_{12} = p_{21}$. Exact cluster recovery is notably more difficult in the region $(p_{12}, p_{21}) \approx (0.5, 1)$ compared to $(p_{12}, p_{21}) \approx (0.5, 0)$, a similar conclusion holds on the axis perpendicular to it. This can also be seen in the feasibility region. Similar shape can also be confirmed in the north-eastern corner where a yellow triangular area is visible in both plots. From the blue line in Figure 3.7a The most difficult region to cluster appears to be the diagonal.

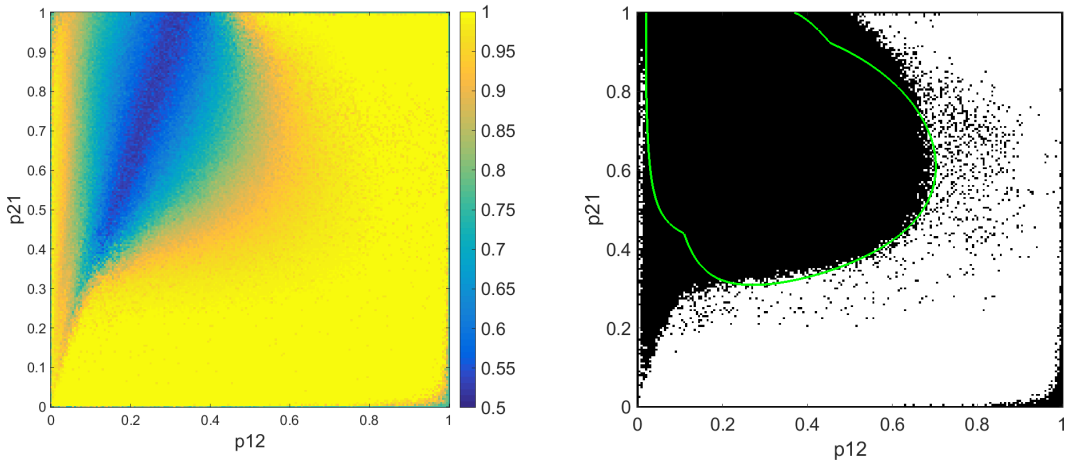
It can be noted that if ϵ is increased from 0.05 to 0.025, $\hat{\mathcal{F}}$ in Figure 3.7a will grow towards the sides, away from the diagonal $p_{12} = p_{21}$. This suggests that the figure tends to the feasibility region instead of expanding more on what appears to be the difficult parameters on the diagonal. When we compare the estimated feasibility region and the feasibility region in Figure 3.7b we see that the difference is in the extension of the diagonal along the difficult line.



(a) Accuracy ν (b) $\hat{\mathcal{F}}(0.027)$

Figure 3.7: The accuracy when $\alpha_2 = 0.5$. On the left the expectation of ν that is estimated using 10 trials for every rasterpoint $(p_{12}, p_{21}) \in (0, 1)^2$. On the right the estimated feasibility region for $\epsilon = 0.027$ is shown.

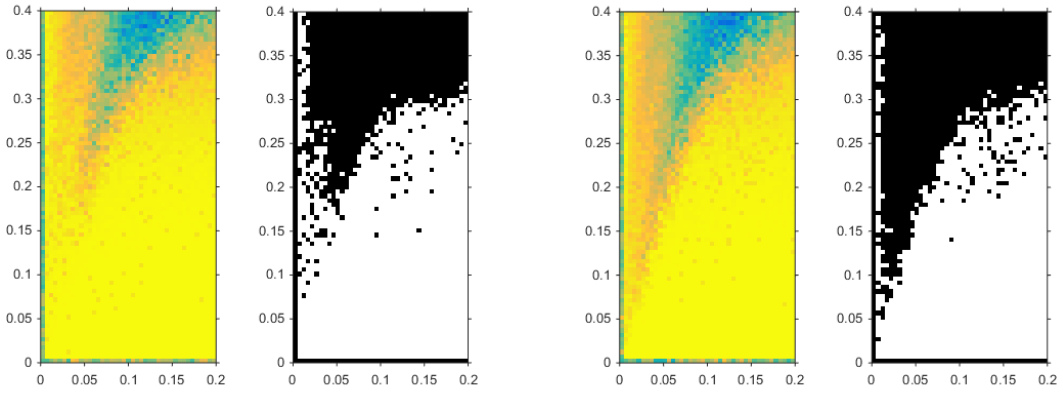
Next we repeat this simulation in case the fraction of states α_2 is set to 0.25. This yields Figure 3.8. From the feasibility regions in Figure 3.3 and the estimated feasibility regions in Figure 3.7b and Figure 3.8b, we can see similarities in the behaviour of the feasibility region shape. There is a shift in the feasibility region to the north-west section when changing α_2 from 0.5 to 0.25 which is the same behaviour as seen in the estimated feasibility region. If we take a closer look at Figure 3.8a similar conclusions can be drawn as for the case of Figure 3.7a. There appears to be a line where clustering is difficult, based on both estimated feasibility regions this seems to be $p_{21} \approx p_{12}(1-\alpha_2)/(\alpha_2)$. Around this line the biggest difference between the estimated feasibility region and feasibility region can be noted. Also in the case of $\alpha_2 = 0.25$ the estimated feasibility regions seems to expand more towards the sides than along the difficult line.



(a) ν (b) $\hat{\mathcal{F}}(0.027)$

Figure 3.8: The accuracy when $\alpha_2 = 0.25$. On the left the expectation of ν that is estimated using 10 trials for every rasterpoint $(p_{12}, p_{21}) \in (0,1)^2$. On the right the estimated feasibility region for $\epsilon = 0.027$ is shown.

Besides this difficult parameter area around this line there is another small triangular region in the bottom west corner. This unexpected behaviour in Figure 3.8 is in the region $p_{12} \in (0, 0.2)$ and $p_{21} \in (0, 0.4)$. In this region it seems harder for the CA to cluster than we expect from the feasibility region. This behaviour can be partly accounted for with the number of times the improvement step is ran, that is 6 times. When we rerun this analysis with an increased number of improvement steps, 16, we obtain Figure 3.9a. For comparison the same region is plotted in Figure 3.9b when the number of improvement steps was 6. We can see that the performance of the clustering algorithm increased since the feasibility region became larger. However, there is still a portion that falls outside the feasibility region. This portion is larger than expected from the theoretical analysis. This region is partly on the difficult line and similar behaviour can also be seen in Figure 3.7b for the feasibility region with $a_2 = 0.5$. Increasing the number of improvement steps improved the performance of the algorithm but it did not yield the exact theoretical feasibility region as seen in Figure 3.3a.



(a) ν and $\hat{\mathcal{F}}(0.027)$ with 16 improvement steps (b) ν and $\hat{\mathcal{F}}(0.027)$ with 6 improvement steps

Figure 3.9: Accuracy and estimated feasibility region for $\alpha_2 = 0.25$. In both figures ν and $\hat{\mathcal{F}}(0.027)$ are averaged over 10 trials, for values $p_{12} \in (0, 0.2)$ and $p_{21} \in (0, 0.4)$. This simulation is done with 16 improvement steps (left) and 6 improvement steps (right).

In Figure 3.7 and Figure 3.8 the estimated feasibility region do not match the feasibility region perfectly. The number of states however was set to a mere 300. We therefore examine the influence of the number of states n on the results. Such investigation is warranted because the exact recovery

region is defined asymptotically, i.e., when $n \to \infty$. Let us fix the parameters $\alpha_2 = 0.5$ and $p_{12} = p_{21}$. The accuracy v is plotted on the y-axis for the number of states $n = (300, 700, 1000)$ in Figure 3.10. The length of the trajectory is on the critical regime of $T = n \log n$. For the given parameters we can calculate the transition probabilities where the information quantity is equal to one. This probabilities are the exact points where we expect the BMC to transfer from feasible to infeasible to cluster. Solving $I(\alpha, p) = 1$ using (3.4) on the diagonal of the feasibility region with $\alpha_2 = 0.5$ reduces the expression $I(\alpha_2 = 0.5, p = p_{12} = p_{21}) = 1$ to

$$2(1-p)\log\frac{1-p}{p} + 2p\log\frac{p}{1-p} = 1\,.$$

The solutions to this equation are $p \approx 0.26045$ and $p \approx 0.73955$. The horizontal green lines in Fig. 3.10 are the boundary probabilities where $I(\alpha, p) = 1$ for the given parameters ($\alpha_2 = 0.5$ and $p_{12} = p_{21}$).

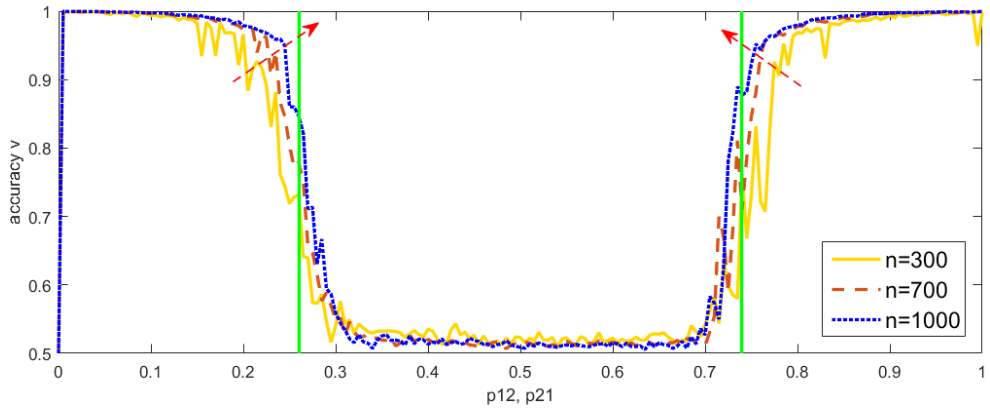


Figure 3.10: Sample mean accuracy of the CA for various $n = (300, 700, 1000)$ with $T = n \log n$ and α_2 is 0.5. Every sample point is the average of 10 trials. The vertical green lines denote the $p = p_{12} = p_{21}$ where $I(\alpha, p) = 1$.

In Figure 3.10 the diagonal ($p_{12} = p_{21}$) of the accuracy is shown for various different choices of the number of states. When you compare this with the $I(\alpha, p) = 1$ cuts, it becomes clear that, if the number of states increases, the accuracy tends to rise and starts dropping when you get closer to this bound with increasing n. This is in line with the asymptotic behaviour you expect. Other than the trend towards a higher accuracy, the line seems to be more smooth which suggests more constant performance of the CA when the number of states goes up.

3.4.3 | Conclusion

We finally answer the research question of this section:

> What is the performance of the Clustering Algorithm in the critical regime, and is this performance adequate for implementation in a BMC-MDP approach?

To answer this question we must realise that the examples provided are in the simple case of 2 clusters. For this small case there is a quite large estimated region where exact recovery is not possible when the trajectory length is set to $T = n \log n$. This estimated region matches the expected theoretical region. The estimated feasibility region seems to match the feasibility region more when the number of states rises based on Figure 3.10. From the estimated feasibility regions and the feasibility regions we conclude that the region $I(\alpha, p) > 1$ describes the region where asymptotic exact recovery is possible if the trajectory length is $n \log n$. It has to be noted that the only real difference between the estimated feasibility region and the feasibility regions appears to be around the line $p_{21} \approx p_{12}(1 - \alpha_2)/\alpha_2$.

For general use in a BMC–MDP approach it seems better to operate in to the dense regime where asymptotic exact recovery is possible for any $I(\alpha, p) > 0$. If there is more known a priori about the problem and you know approximately what kind of $I(\alpha, p)$ you are dealing with, it will be beneficial for the overall BMC–MDP performance to shorten the clustering trajectory length. For example, if you have a problem where you know a priori what a lower bound to $I(\alpha, p)$ is, it is possible to adjust the clustering length accordingly: the theoretically feasibility region is defined via $I(\alpha, p) > 1/c$ where the trajectory length $T = cn \log n$ with $c \in (0, \infty)$ [40].

3.5 | Behaviour of a mixed Markov Process

In a general MDP we potentially have a different transition matrix depending on the action taken as seen in Section 2.4. In this section we investigate the behaviour of a Markov chain that is caused by alternating the transition matrix. During the initial phase, in a MDP, when the states are not clustered yet such kind of Markov chain may be expected. Because the CA is proven to be optimal when the input is a trajectory generated by a single block transition matrix, the objective of this section will be to determine if the stochastic behaviour of a Markov trajectory when the dynamics are governed by multiple transition matrices is close to the stochastic behaviour of a Markov chain that has a single transition matrix underlying its dynamic .

In Section 3.5.1 we define the mixed and pure Markov process that we are going to investigate. In Sections 3.5.2 - 3.5.4 we investigate these two processes using notions of difference between stochastic processes. Section 3.5.5 summarizes our findings.

3.5.1 | Definition of the mixed Markov chain

We start by defining the process λ as a Markov chain with a single transition matrix

$$B \triangleq \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,n} \\ b_{2,1} & b_{2,2} & \dots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \dots & b_{n,n} \end{bmatrix}.$$

This matrix defines the probability of transitioning between states. The probability of transition is independent of the action as there is only a single transition matrix and is given by

$$\mathbb{P}_\lambda(S_{t+1} = i' \mid S_t = i) = b_{i,i'} \quad \text{for all } i, i' \in \mathcal{S}. \tag{3.9}$$

For (3.9) to be a distribution the rows of the matrix have to sum to one because every row contains the probability of the next state so $\sum_{k=1}^{n} b_{i,k} = 1$ for all states $i \in \mathcal{S}$.

We define the event of seeing a specific state trajectory by $s_{[T]} \triangleq \{S_0 = s_0, S_1 = s_1, \dots, S_T = s_T\}$. The probability of this event under the process λ is given by

$$\mathbb{P}_\lambda(s_{[T]}) = \mathbb{P}_\lambda(S_0 = s_0) \prod_{t=0}^{T-1} \mathbb{P}_\lambda(S_t = s_t \mid S_{t-1} = s_{t-1}) = \mathbb{P}_\lambda(S_0 = s_0) \prod_{t=0}^{T-1} b_{s_t, s_{t+1}}. \tag{3.10}$$

Here $\mathbb{P}_\lambda(S_0 = s_0)\ \forall s_0 \in \mathcal{S}$ denotes the initial distribution. It describes the probability of the starting state of the trajectory given the process λ.

Next, we define the mixed Markov chain μ. The output of this process is a state trajectory $s_{[T]}$. The next state S_{t+1} in this trajectory is determined by the current state $S_t \in \mathcal{S} = \{1, \dots, n\}$ and the current action $A_t \in \mathcal{A} = \{1, \dots, l\}$. The probability of transitioning to a next state S_{t+1}, given the current state S_t and action A_t, is given by

$$\mathbb{P}_\mu(S_{t+1} = i' \mid S_t = i, A_t = j) = d_{i,i'}^{[j]} \quad \text{for all } i, i' \in \mathcal{S} \text{ and } j \in \mathcal{A}.$$

All of these entries can be formulated in l (number of actions) matrices

$$D^{[j]} \triangleq \begin{bmatrix} d_{1,1}^{[j]} & d_{1,2}^{[j]} & \dots & d_{1,n}^{[j]} \\ d_{2,1}^{[j]} & d_{2,2}^{[j]} & \dots & d_{2,n}^{[j]} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n,1}^{[j]} & d_{n,2}^{[j]} & \dots & d_{n,n}^{[j]} \end{bmatrix} \text{ for all } j \in \mathcal{A}.$$

The rows of every matrix have to sum to one because every row contains the probability of the next state so $\sum_{k=1}^{n} d_{i,k}^{[j]} = 1$ for all actions $j \in \mathcal{A}$ and states $i \in \mathcal{S}$.

The difference between process μ and process λ is that there is an underlying action trajectory $\{A_0, \dots, A_{T-1}\}$ that influences the state trajectory. The current action A_t determines the transition matrix that is picked for transitioning towards the next state. To fully specify the process μ we now define a probability of picking the action A_t on decision epoch t given the process μ and the history $\mathcal{H}_t \triangleq \{S_t =$

$s_t, \dots, S_0 = s_0, A_{t-1} = a_{t-1}, \dots, A_0 = a_0\}$ up to that decision epoch. This probability can depend on the current and previous states as well as the previous actions:

$$\mathbb{P}_\mu(A_t = j_t \mid \mathcal{H}_t) = \mathbb{P}_\mu(A_t = j_t \mid S_t = i_t, S_{t-1} = i_{t-1}, \dots, S_0 = i_0, A_{t-1} = j_{t-1}, \dots, A_0 = j_0)\,. \tag{3.11}$$

Similar to the event $s_{[T]}$ we define the event of seeing a specific action sequence by $a_{[T]} \triangleq \{A_0 = a_0, A_1 = a_1, \dots, A_T = a_T\}$ with every $a_t \in \mathcal{A}$.

The probability of event $s_{[T]}$ under μ is given by

$$\mathbb{P}_\mu(s_{[T]}) = \sum_{a_{[T-1]} \in U_{T-1}} \mathbb{P}_\mu(s_{[T]}, a_{[T-1]})\,, \tag{3.12}$$

i.e., the summation over the probability of all possible action trajectories $a_{[T]} \in U_{T-1}$. Here U_{T-1} is the set of all possible action trajectories of length $T-1$.

The probability of a combined state and action trajectory under the process μ is given by

$$\begin{aligned}\mathbb{P}_\mu(s_{[T]}, a_{[T-1]}) &= \mathbb{P}_\mu(S_0 = s_0) \prod_{t=0}^{T-1} \Big(\mathbb{P}_\mu(A_t = a_t \mid \mathcal{H}_t)\mathbb{P}_\mu(S_{t+1} = s_{t+1} \mid S_t = s_t, A_t = a_t)\Big) \\ &= \mathbb{P}_\mu(S_0 = s_0) \prod_{t=0}^{T-1} \Big(\mathbb{P}_\mu(A_t = a_t \mid \mathcal{H}_t) d_{s_t, s_{t+1}}^{[a_t]}\Big)\,.\end{aligned} \tag{3.13}$$

Here $\mathbb{P}_\mu(S_0 = s_0)\ \forall s_0 \in \mathcal{S}$ is the initial state distribution and denotes the probability of the starting state of the trajectory given the process μ.

3.5.2 | Stationary distribution

Now that the different processes λ and μ are defined we are going to look at the equilibrium probability of both procceses. When the equilibrium distributions are not equal, we cannot expect the processes to be similar. To achieve this goal we consider the case that there are $n = 2$ states and $l = 2$ actions. The transition matrices of the mixed Markov process μ are given by

$$D^{[1]} \triangleq \begin{bmatrix} d_{11}^{[1]} & d_{12}^{[1]} \\ d_{21}^{[1]} & d_{22}^{[1]} \end{bmatrix} \qquad \text{and} \qquad D^{[2]} \triangleq D^{[1]} + \epsilon \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

Because every entry of this matrix is a transition probability every $d_{ik}^{[j]} \in (0, 1)$. For this specific configuration this gives bounds on the possible value of ϵ, i.e., $\max(-d_{11}, -d_{22}) < \epsilon < \min(1 - d_{11}, 1 - d_{22})$.

We assume that the Markov walk started from equilibrium. We also assume that the probability of picking action one or two is independent of the current state and the time t and thus independent of the history of the states and actions. This assumption implies that we can describe the probability of picking an action by a single probability. From here on out we use the following notation for the probabilities of taking an action $\mathbb{P}(A_t)$ and the probability being in a state $\mathbb{P}S_t)$ at time t:

$$\begin{aligned} \mathbb{P}(A_t = 1) &= k_1\,, & \mathbb{P}(S_t = 1) &= \Pi_1\,, \\ \mathbb{P}(A_t = 2) &= k_2\,, & \mathbb{P}(S_t = 2) &= \Pi_2\,. \end{aligned}$$

Given that the number of states and actions is two it follows that $k_2 = 1 - k_1$ and $\Pi_1 = 1 - \Pi_2$.

We can now analyse the equilibrium behaviour for the mixed Markov chain, which satisfies these local balance equations:

$$[\Pi_1, \Pi_2] \triangleq \mathbf{\Pi} = \mathbf{\Pi} k_1 \bar{D}^{[1]} + \mathbf{\Pi} k_2 \bar{D}^{[2]}\,. \tag{3.14}$$

Because of the defined structure of $D^{[1]}$ and $D^{[2]}$, we ease notation and use $d_{ij}^{[1]} = d_{ij}$ in the rest of this example. Expanding (3.14) component wise, we find

$$\begin{aligned} \Pi_1 &= \Pi_1 k_1 d_{11} + \Pi_2 k_1 (1 - d_{22}) + \Pi_1 k_2 (d_{11} + \epsilon) + \Pi_2 k_2 (1 - d_{22} - \epsilon)\,, \\ \Pi_2 &= \Pi_1 k_1 (1 - d_{11}) + \Pi_2 k_1 d_{22} + \Pi_1 k_2 (1 - d_{11} - \epsilon) + \Pi_2 k_2 (d_{22} + \epsilon)\,. \end{aligned} \tag{3.15}$$

Additionally with $\Pi_1 + \Pi_2 = 1$, we can solve (3.15) which gives

$$\Pi_1 = \frac{(1-d_{22}) - k_2\epsilon}{(1-d_{22}) - k_2\epsilon + 1 - d_{11} - k_2\epsilon} \quad ,\Pi_2 \quad = \frac{1 - d_{11} - k_2\epsilon}{(1-d_{22}) - k_2\epsilon + 1 - d_{11} - k_2\epsilon}.$$

This results in the cluster equilibrium distribution of

$$\mathbf{\Pi}_\mu = \left[\frac{1-d_{22}-k_2\epsilon}{1-d_{22}-k_2\epsilon+1-d_{11}-k_2\epsilon}, \frac{1-d_{11}-k_2\epsilon}{1-d_{22}-k_2\epsilon+1-d_{11}-k_2\epsilon}\right].$$

Next we look at the equilibrium distribution of the pure Markov chain with transition matrix

$$\bar{B} = \begin{bmatrix} 1-b_{12} & b_{12} \\ b_{21} & 1-b_{21} \end{bmatrix}.$$

Solving the local balance equations, we find the equilibrium distribution

$$\mathbf{\Pi}_B = \left[\frac{b_{21}}{b_{12}+b_{21}}, \frac{b_{12}}{b_{12}+b_{21}}\right].$$

If we take the process λ to be a pure BMC with transition matrix $\tilde{B} = k_1 * D^{[1]} + k_2 * D^{[2]}$, this gives

$$\tilde{B} = \begin{bmatrix} d_{11}+k_2\epsilon & 1-d_{11}-k_2\epsilon \\ 1-d_{22}-k_2\epsilon & d_{22}+k_2\epsilon \end{bmatrix},$$

and its equilibrium distribution will be given by

$$\mathbf{\Pi}_\lambda = \left[\frac{1-d_{22}-k_2\epsilon}{1-d_{22}-k_2\epsilon+1-d_{11}-k_2\epsilon}, \frac{1-d_{11}-k_2\epsilon}{1-d_{22}-k_2\epsilon+1-d_{11}-k_2\epsilon}\right].$$

This is the same equilibrium distribution as the equilibrium distribution spanned by the process of an mixed Markov chain with fixed probabilities of picking one of two transition matrices. From this analysis we can conclude that there exist a pure Markov transition matrix that has identical equilibrium behaviour as the mixed Markov process.

3.5.3 | Total Variation Distance

We are now going to investigate these two processes using a general statistical distance measure of two distribution. Specifically, we use the Total Variation Distance (TVD) between the two distributions. This distance is defined in term of events and is related to the L^1 norm if the set is countable [33, proposition 4.2]. The TVD is defined as the summed difference of the probabilities of single events between two random variables as

$$D_{tv}(\mu,\lambda) \triangleq \frac{1}{2}\sum_{E\in\Sigma}\left|\mathbb{P}_\mu(E) - \mathbb{P}_\lambda(E)\right|. \tag{3.16}$$

Here the set Σ denotes the set of all possible single events of the random variables μ and λ. To compare the two processes we must formulate the process λ on the same state space as the process μ. We do this by defining l transition matrices $B^{[j]}$ similar to $A^{[j]}$ with

$$B^{[j]} = B \text{ for all } j \in \mathcal{A}.$$

For the alternated process λ we also define the probability of selection action $\mathbb{P}_\lambda(A_t = j \mid \mathcal{H}_T)$ for all $j \in \mathcal{A}$ depending on the history $\mathcal{H}_T$, identical to (3.11). The probability of both the action and state trajectory are identical to (3.13) but with the matrix $b^{[j]}$ instead of $d^{[j]}$. From here on out for the rest of this Section 3.5.3 the λ process refers to this alternated version of the λ process with multiple transition matrices.

Our objective is to find the matrix B^* that minimizes the TVD of these two processes. The set Σ denotes the set of all events $s_{[T]}$ and $a_{[T-1]}$. The set of all action trajectories $a_{[T-1]}$ is denoted by U_{T-1}. We can rewrite (3.16) to

$$D_{tv}(\mu,\lambda) = \frac{1}{2}\sum_{s_{[T]},a_{[T-1]}\in\Sigma}\left|\mathbb{P}_\mu(s_{[T]},a_{[T-1]}) - \mathbb{P}_\lambda(s_{[T]},a_{[T-1]})\right|. \tag{3.17}$$

The matrix B^* we are going to derive is the matrix that lower bounds this TVD. We will derive this matrix B^* under Assumption 1.

Assumption 1. *For the process $K = \{\mu, \lambda\}$ we assume that:*

- *The probability of picking an action at time step t only depends on the current state $S_t \in \mathcal{S}$ and is independent of the time step. This reduces $\mathbb{P}_K(A_t = a_t \mid \mathcal{H}_t)$, (3.11), to $\mathbb{P}_K(A_t = a_t \mid S_t = s_t) = \mathbb{P}_K(A = a_t \mid S = s_t)$ for K both μ and λ.*
- *The initial state distribution is the same for both processes, i.e., $\mathbb{P}_\lambda(S_0 = i) = \mathbb{P}_\mu(S_0 = i)$ for all $i \in \mathcal{S}$.*

The matrix B^* that minimizes a lower bound on the TVD is given in Proposition 1.

Proposition 1. *Under Assumption 1, if the action $j \in \mathcal{A}$ is selected with a fixed probability given the current state $S_t = i$ independent of t, then the transition matrix B^* that minimizes a lower bound on the Total Variation Distance is*

$$b^*_{i,i'} = \sum_{j \in \mathcal{A}} \mathbb{P}_\mu(A = j \mid S = i) d^{[j]}_{i,i'} \text{ for all } i, i' \in \mathcal{S}\,.$$

To prove Proposition 1, we will first introduce and prove Lemma 1. This lemma is used in the analysis of the minimum bound and enables us to rewrite the equation so that we can find the matrix B^* that minimizes our lower bound.

Lemma 1. *Under Assumption 1, if the probability of selecting action $A_t = j$ at time step t is independent of the time epoch and only depends on the current state $S_t = i$ of the state trajectory we get $\mathbb{P}(A_t = j \mid S_t = i) = \mathbb{P}(A = j \mid S = i)$, with $j \in \mathcal{A}$ so that $\sum_{j=1,\dots,l} \mathbb{P}(A = j \mid S = i) = 1$, for any $i \in \mathcal{S}$. Then*

$$\sum_{a_{[T-1]} \in U_{T-1}} \prod_{t=0}^{T} \mathbb{P}(A = a_t \mid S = s_t) d^{[a_t]}_{s_t, s_{t+1}} = \prod_{t=0}^{T} \sum_{k \in \mathcal{A}} \mathbb{P}(A = k \mid S = s_t) d^{[k]}_{s_t, s_{t+1}}\,.$$

Proof. First we expand the sum of the action trajectory on the left-hand side, this results in

$$\sum_{a_{[T-1]} \in U_{T-1}} \prod_{t=0}^{T} \mathbb{P}(A = a_t \mid S = s_t) d^{[a_t]}_{s_t, s_{t+1}} = \sum_{a_0=1}^{l} \dots \sum_{a_T=1}^{l} \prod_{t=0}^{T} \mathbb{P}(A = a_t \mid S = s_t) d^{[a_t]}_{s_t, s_{t+1}}\,.$$

Now we set $\mathbb{P}(A = a_t \mid S = s_t) \triangleq k^{s_t}_{a_t}$, such that

$$\sum_{a_0=1}^{l} \dots \sum_{a_T=1}^{l} \prod_{t=0}^{T} \mathbb{P}(A = a_t \mid S = s_t) d^{[a_t]}_{s_t, s_{t+1}} = \sum_{a_0=1}^{l} \dots \sum_{a_T=1}^{l} k^{s_0}_{a_0} d^{[a_0]}_{s_0, s_1} k^{s_1}_{a_1} d^{[a_1]}_{s_1, s_2} \dots k^{s_T}_{a_T} d^{[a_T]}_{s_T, s_{T+1}}\,.$$

Because each term $k^{s_t}_{a_t} d^{[a_t]}_{s_t s_{t+1}}$ is independent of the other iterands $a_i, i \neq t$, we have

$$\sum_{a_0=1}^{l} \dots \sum_{a_T=1}^{l} k^{s_0}_{a_0} d^{[a_0]}_{s_0, s_1} k^{s_1}_{a_1} d^{[a_1]}_{s_1, s_2} \dots k^{s_T}_{a_T} d^{[a_T]}_{s_T, s_{T+1}} = \sum_{a_0=1}^{l} k^{s_0}_{a_0} d^{[a_0]}_{s_0, s_1} \sum_{a_1=1}^{l} k^{s_1}_{a_1} d^{[a_1]}_{s_1, s_2} \dots \sum_{a_T=1}^{l} k^{s_T}_{a_T} d^{[a_T]}_{s_T, s_{T+1}}\,.$$

Filling in $k^{s_t}_{a_t}$ and rewriting this equation gives

$$\sum_{a_{[T-1]} \in U_{T-1}} \prod_{t=0}^{T} \mathbb{P}(A = a_t \mid S = s_t) d^{[a_t]}_{s_t, s_{t+1}} = \prod_{t=0}^{T} \sum_{a_t=1}^{l} \mathbb{P}(A = a_t \mid S = s_t) d^{[a_t]}_{s_t, s_{t+1}}\,.$$

This completes the proof. □

Now we are going to show Proposition 1. We use (3.13) for both processes μ and λ to rewrite (3.17) and find

$$D_{tv}(\mu, \lambda) = \frac{1}{2} \sum_{s_0=1}^{n} \sum_{s_1=1}^{n} \dots \sum_{s_T=1}^{n} \sum_{a_0=1}^{l} \sum_{a_1=1}^{l} \dots \sum_{a_{T-1}=1}^{l} \left| \mathbb{P}_\mu(S_0 = s_0) \prod_{t=0}^{T-1} \mathbb{P}_\mu(A_t = a_t \mid \mathcal{H}_t) d^{[a_t]}_{s_t, s_{t+1}} - \mathbb{P}_\lambda(S_0 = s_0) \prod_{t=0}^{T-1} \mathbb{P}_\lambda(A_t = a_t \mid \mathcal{H}_t) b^{[a_t]}_{s_t, s_{t+1}} \right|. \quad (3.18)$$

We apply Assumption 1 and take the initial state distribution outside the absolute brackets. This gives:

$$= \frac{1}{2} \sum_{s_0=1}^{n} \mathbb{P}(S_0 = s_0) \sum_{s_1=1}^{n} \dots \sum_{s_T=1}^{n} \sum_{a_0=1}^{l} \sum_{a_1=1}^{l} \dots \sum_{a_{T-1}=1}^{l} \left| \prod_{t=0}^{T-1} \mathbb{P}_\mu(A = a_t \mid S = s_t) d_{s_t,s_{t+1}}^{[a_t]} - \prod_{t=0}^{T-1} \mathbb{P}_\lambda(A = a_t \mid S = s_t) b_{s_t,s_{t+1}}^{[a_t]} \right|.$$

Next we use the triangle inequality $|x + y| \leq |x| + |y|$, this results in a lower bound on the TVD. This lower bound is given by

$$D_{tv}(\mu, \lambda) \geq \frac{1}{2} \sum_{s_0=1}^{n} \mathbb{P}(S_0 = s_0) \sum_{s_1=1}^{n} \dots \sum_{s_T=1}^{n} \left| \sum_{a_0=1}^{l} \sum_{a_1=1}^{l} \dots \sum_{a_{T-1}=1}^{l} \left(\prod_{t=0}^{T-1} \mathbb{P}_\mu(A = a_t \mid S = s_t) d_{s_t,s_{t+1}}^{[a_t]} - \prod_{t=0}^{T-1} \mathbb{P}_\lambda(A = a_t \mid S = s_t) b_{s_t,s_{t+1}}^{[a_t]} \right) \right|.$$

Now we examine the term within the absolute brackets and we assume the trajectory $s_{[T]}$ to be given. If the term inside the absolute brackets is zero for any trajectory, the sum over the set of all state trajectories will also be zero. Therefore we focus on the term

$$\left| \sum_{a_{[T-1]} \in U_{T-1}} \left(\prod_{t=0}^{T-1} \mathbb{P}_\mu(A = a_t \mid S = s_t) d_{s_t,s_{t+1}}^{[a_t]} - \prod_{t=0}^{T-1} \mathbb{P}_\lambda(A = a_t \mid S = s_t) b_{s_t,s_{t+1}}^{[a_t]} \right) \right|. \tag{3.19}$$

We assume T to be finite so we can use the finite property of a sum sequence $\sum(a - b) = \sum(a) - \sum(b)$ to rewrite (3.19) to obtain

$$\left| \sum_{a_{[T-1]} \in U_{T-1}} \prod_{t=0}^{T-1} \mathbb{P}_\mu(A = a_t \mid S = s_t) d_{s_t,s_{t+1}}^{[a_t]} - \sum_{a_{[T-1]} \in U_{T-1}} \prod_{t=0}^{T-1} \mathbb{P}_\lambda(A = a_t \mid S = s_t) b_{s_t,s_{t+1}}^{[a_t]} \right|.$$

Because of Assumption 1 we can now use Lemma 1 to get:

$$\left| \prod_{t=0}^{T-1} \sum_{j \in \mathcal{A}} \mathbb{P}_\mu(A = j \mid S = s_t) d_{s_t,s_{t+1}}^{[j]} - \prod_{t=0}^{T-1} \sum_{j \in \mathcal{A}} \mathbb{P}_\lambda(A = j \mid S = s_t) b_{s_t,s_{t+1}}^{[j]} \right|$$

$$= \left| \prod_{t=0}^{T-1} \sum_{j \in \mathcal{A}} \mathbb{P}_\mu(A = j \mid S = s_t) d_{s_t,s_{t+1}}^{[j]} - \prod_{t=0}^{T-1} b_{s_t,s_{t+1}} \right|.$$

This shows that the lower bound on the TVD is zero for any s_t, s_{t+1} if the matrix entries are

$$b_{s_t,s_{t+1}}^{*} = \sum_{j \in \mathcal{A}} \mathbb{P}_\mu(A = j \mid S = s_t) d_{s_t,s_{t+1}}^{[j]} \text{ for all } s_t, s_{t+1} \in \mathcal{S}. \tag{3.20}$$

We have now shown that Proposition 1 holds and that for (3.20) we have a minimum bound on $D_{tv}(\mu, \lambda)$ of zero. This does not imply that this is the matrix B^* that minimises the original equation. It gives one matrix B^* that results in a $D_{tv}(\mu, \lambda)$ that is potentially zero.

3.5.4 | Summed difference in probabilities of trajectory

Following the minimum bound on the Total Variation Distance we now investigate the summed difference of all events $s_{[T]}$ under μ and λ. This investigation is warranted because clustering is done based

on a single trajectory. If the probability of a trajectory on both processes is equal then this would imply that the input of the clustering algorithm would be identical with equal probability. Hence the two processes are identical from a clustering perspective in that scenario.

In this section the process λ denotes a pure Markov chain with a transition matrix B as defined in Section 3.5.1. Here μ denotes the mixed process of several transition matrices $D^{[a]}$, the choice of the transition matrix depends on the action A_t chosen at the decision epoch t. Let X_T denote the set of all possible state trajectories $s_{[T]}$. The objective is to find the transition matrix B^* that minimises the summed difference in probability over all possible finite size trajectory $s_{[T]}$ of length T. The summed difference in probability is defined as

$$\sum_{s_{[T]} \in X_T} \left| \mathbb{P}_\mu(s_{[T]}) - \mathbb{P}_\lambda(s_{[T]}) \right| . \tag{3.21}$$

Using (3.12) we can rewrite (3.21) and find

$$\sum_{s_{[T]} \in X_T} \left| \mathbb{P}_\mu(s_{[T]}) - \mathbb{P}_\lambda(s_{[T]}) \right| = \sum_{s_0=1}^{n} \sum_{s_1=1}^{n} \dots \sum_{s_T=1}^{n} \left| \sum_{a_0=1}^{l} \sum_{a_1=1}^{l} \dots \sum_{a_{T-1}=1}^{l} \mathbb{P}_\mu(s_{[T]}, a_{[T-1]}) - \mathbb{P}_\lambda(s_{[T]}) \right| . \tag{3.22}$$

Similar to the previous section we are going to show that a matrix B^* that minimizes (3.22) is

$$b^*_{i,i'} = \sum_{j \in \mathcal{A}} \mathbb{P}(A_t = j \mid S_t = i) d^{[j]}_{i,i'} \text{ for all } i, i' \in \mathcal{S} . \tag{3.23}$$

We do this under the assumption that picking the actions only depends on the current state and is done independent of time. Also the initial state distribution for both processes are identical. In other words Assumption 1 holds in this analysis. The first assumption is based on a random choice of actions prior to the clustering step by some distribution. We start by substituting (3.10) and (3.12) and this gives

$$\sum_{s_{[T]} \in X_T} \left| \mathbb{P}_\mu(s_{[T]}) - \mathbb{P}_\lambda(s_{[T]}) \right| =$$

$$\sum_{s_{[T]} \in X_T} \left| \sum_{a_{[T-1]} \in J_{T-1}} \mathbb{P}_\mu(S_0 = s_0) \prod_{t=0}^{T-1} \mathbb{P}_\mu(A_t = a_t \mid \mathcal{H}_t) d^{[a_t]}_{s_t,s_{t+1}} - \mathbb{P}_\lambda(S_0 = s_0) \prod_{t=0}^{T-1} b_{s_t,s_{t+1}} \right| .$$

We use Assumption 1 and rewrite the equation into

$$\sum_{s_{[T]} \in X_T} \left| \mathbb{P}_\mu(s_{[T]}) - \mathbb{P}_\lambda(s_{[T]}) \right| = \sum_{s_{[T]} \in X_T} \mathbb{P}(S_0 = s_0) \left| \sum_{a_{[T-1]} \in J_{T-1}} \prod_{t=0}^{T-1} \mathbb{P}_\mu(A_t = a_t \mid \mathcal{H}_t) d^{[a_t]}_{s_t,s_{t+1}} - \prod_{t=0}^{T-1} b_{s_t,s_{t+1}} \right| .$$

The summed difference of all trajectory probabilities is equal to zero if the following term is zero for all $s_{[T]} \in X_T$:

$$\left| \sum_{a_{[T-1]} \in U_{T-1}} \prod_{t=0}^{T-1} \mathbb{P}_\mu(A_t = a_t \mid S_t = s_t) d^{[a_t]}_{s_t,s_{t+1}} - \prod_{t=0}^{T-1} b_{s_t,s_{t+1}} \right| = 0 . \tag{3.24}$$

Now we apply Lemma 1. We use this lemma to swap the sum and product that yields

$$\left| \prod_{t=0}^{T-1} \sum_{a_t \in \mathcal{A}} \mathbb{P}(A_t = a_t \mid S_t = s_t) d^{[a_t]}_{s_t,s_{t+1}} - \prod_{t=0}^{T-1} b_{s_t,s_{t+1}} \right| = 0 .$$

This equation is zero if the two terms are equal for all operands, i.e.,

$$\sum_{a_t \in \mathcal{A}} \mathbb{P}(A_t = a_t \mid S_t = s_t) d^{[a_t]}_{s_t,s_{t+1}} = b_{s_t,s_{t+1}} \text{ for all } s_t, s_{t+1} \in \mathcal{S} .$$

This shows that a transition matrix B^* for the process λ that satisfies (3.24) is element–wise given by

$$b^*_{i,i'} = \sum_{j \in \mathcal{A}} \mathbb{P}(A_t = j \mid S_t = i) d^{[j]}_{i,i'} \text{ for all } i, i' \in \mathcal{S}\,. \tag{3.25}$$

If the probability for action $j \in \mathcal{A}$ is independent of the current state, namely $\mathbb{P}(A_t = j \mid S = i) = \mathbb{P}(A_t = j \mid S_t = i') = \mathbb{P}(A_t = j)$ for all $i, i' \in \mathcal{S}$, then the transition matrix B^* can be written as

$$B^* = \sum_{j \in \mathcal{A}} \mathbb{P}(A_t = j) D^{[j]}\,. \tag{3.26}$$

If we examine (3.24) we conclude that with the averaged transition matrix as calculated by (3.25) the probability of every trajectory is the same for both the mixed Markov chain as well as the pure Markov chain.

3.5.5 | Conclusion

On the basis of the results in this section we conclude that the mixed Markov chain and the averaged pure Markov chain are statistically similar to each other if the action is randomly picked. By the averaged pure Markov chain we mean that the trajectories are generated by a single transition matrix that satisfies Proposition 1. The results in Section 3.5.2 state that the probability of being in a state in the equilibrium situation are identical. Besides that we showed that a lower bound of the Total Variation Distance is zero if the averaged pure Markov chain is compared to the mixed Markov chain in Section 3.5.3. Finally in Section 3.5.4 we showed that the probability of a single trajectory of both these distributions is equal as well under the transition matrix in Proposition 1.

3.6 | Performance of CA on the mixed Markov chain

Based on the mathematical foundation in the previous section. We are now going to investigate the behaviour of the clustering algorithm on a trajectory of a mixed BMC. This research direction is warranted because in a potential cluster-based MDP approach we have to use the clustering algorithm to cluster a trajectory generated by multiple transition matrices.

We conduct this research by first looking at the performance of the clustering algorithm in the case that the mixed Markov process has transition matrices that are similar to one another for every action. Afterwards, we investigate the situation if the transition matrices differ more substantially. On top of that, we investigate whether the number of actions l has any influence on the performance.

By looking at the feasibility region of trajectories generated by an mixed Markov process and a pure Markov process we gain insight in the performance of the Clustering Algorithm. The conclusion drawn in Section 3.5.5 is confirmed: over a substantial number of trials, the CA seems to have identical performance for both the mixed as the pure process. We conclude, from these experiments, that if we have an mixed BMC and the actions are picked with fixed probability given the state, we can cluster based on a state trajectory as long as the information quantity of the averaged Markov transition matrix falls in the feasibility region. The number of actions by itself have no influence on the behaviour.

3.6.1 | Transition matrices with small deviation

For this analysis we restrict ourselves to a BMC that consists of two clusters. We take two clusters for visualisation purposes as the function $I(\alpha, p)$ and the feasibility region can be plotted in $\mathbb{R}^2$ for insight in the expected behaviour. Recall that the transition matrices for a BMC consisting of two clusters can be described by the parameters p_{12}, p_{21}, α_2 and n. We take the fraction of states in cluster two to be $a_2 = 0.25$. For reference the feasibility region for $\alpha_2 = 0.25$ is shown in green in Figure 3.11. For the experiments conducted in this section the transition matrices are generated from the blue and red regions.

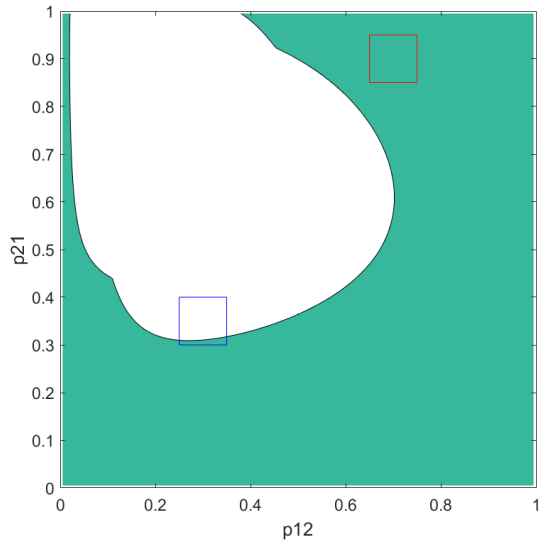


Figure 3.11: Feasibility region $\mathcal{F}$ for $K = 2$ clusters with $\alpha_2 = 0.25$ with $p \in (0,1)^2$ for a trajectory length of $T = n \ln n$.

We generate l transition matrices $P^{(a)} \quad \forall a \in \{1, \dots, l\}$. Every matrix is a intercluster transition matrix describing the probability of moving from a cluster to a cluster. The intercluster transition matrices are uniformly picked from the red region in Figure 3.11 ($p_{12} \in [0.65, 0.75]$ and $p_{21} \in [0.85, 0.95]$) or the blue region ($p_{12} \in [0.25, 0.35]$ and $p_{21} \in [0.3, 0.4]$). At every time step the action, and therefore the transition matrices, have equal probability of being picked: $\mathbb{P}(A_t = i) = 1/l \forall i \in \{1, \dots, l\}$. Based on the findings in Section 3.5 the average matrix $B = \sum_{i=1}^{l} \mathbb{P}(A_t = i) P^{(A_t)}$ is calculated. The trajectory generated by this averaged matrix should have similar clustering performance if the actions are picked according to Assumption 1.

For both processes, the mixed process and the pure process, a trajectory of length $T = n \ln n$ is generated. On these trajectories the clustering algorithm is applied with $\lfloor \ln n \rfloor$ improvement steps. The performance of the CA on both processes is compared using the accuracy in (3.7). We do this experiment for a various number of states and actions. In Table 3.1 the input of these experiments are shown. We can observe the difference in performance in and outside the theoretical feasibility region and we investigate the number of states and actions. The clustering algorithm is ran on 1000 generated

Table 3.1: Input for the accuracy comparison of the Clustering Algorithm for various number of states (n) and actions (l). The trajectory length is $T = n \ln n$ and the number of improvement steps is $\lfloor \ln n \rfloor$.

		Input	
Experiment	Region	Number of states (n)	Number of actions (l)
1	Blue	100	10
2	Blue	100	50
3	Blue	300	50
4	Red	100	10
5	Red	100	50
6	Red	300	50

trajectories for each experiment and the results are visualised using boxplots of the accuracy of these trials. For each experiment the performance of the mixed Markov trajectory and the accuracy of the trajectory generated by a pure transition matrix are shown. The results for all experiments are found in Figure 3.12 and Figure 3.13. The red pluses in these figures are the outliers. These outliers are calculated using the standard Matlab boxplot function.

Based on these figures we can observe that the red region of experiments 4, 5 and 6 appear to be more suitable for clustering compared to blue region of experiment 1,2 and 3. This is in line with the results shown in Figure 3.8 and the position of the red and blue box in respect to the feasibility region. The number of actions on the other hand appear to have no significant influence on the performance.

The lack of difference in results of experiments 1 and 2 and experiments 4 and 5 show this lack of influence. This can be explained by the fact that all matrices are close and the average over ten or fifty action matrices will not influence the trajectory substantially. On the other hand, increasing the number of states yields a different performance. The performance of the clustering algorithm appears to be more consistent as the width of the box in the box plots shrinks in both cases when we compare experiment 2 with 3 and experiment 5 with 6. Furthermore, when we compare the mixed and the pure process we see a negligible difference in accuracy. This indicates that the two processes are equal from this point of view for the clustering algorithm whenever the transition matrices for all actions are similar.

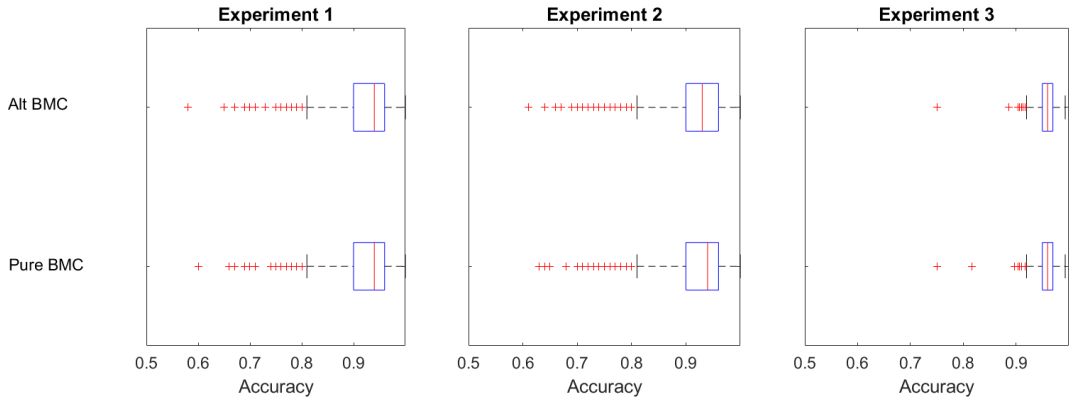


Figure 3.12: Accuracy of clustering algorithm for the mixed Markov process and the pure Markov process for the blue region with the parameters found in Table 3.1. Every boxplot of the accuracy is based on 1000 trials.

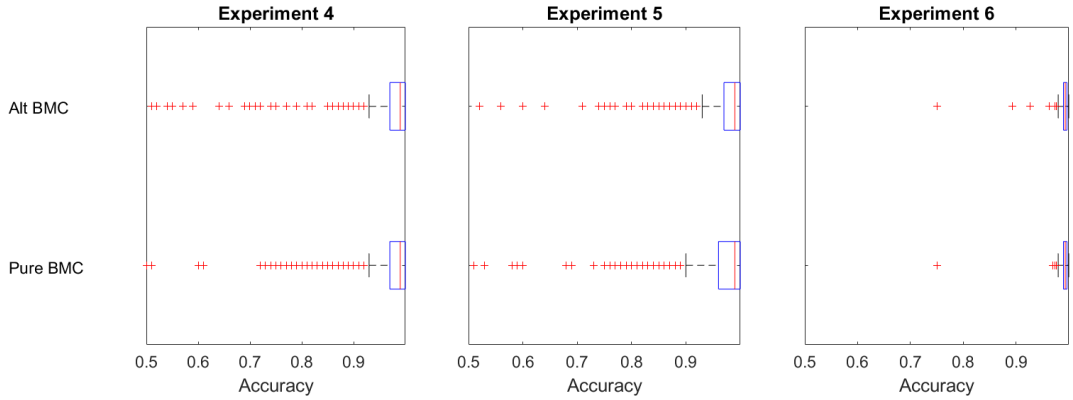


Figure 3.13: Accuracy of clustering algorithm for the mixed Markov process and the pure Markov process for the red region with the parameters found in Table 3.1. Every boxplot of the accuracy is based on 1000 trials.

3.6.2 | Transition matrices that are further apart

We will now investigate the performance of the CA when the transition matrices are spread out over the probability space. The transition matrices are picked from the same regions as before. However, for the alternated process the transition matrix is picked from either the red or blue region in Figure 3.11 with probability a half.

To analyse the performance of the CA in this setting we use the three experimental configurations of the previous section. The trajectory length is $T = n \ln n$ and the number of improvement steps is $\lfloor \ln n \rfloor$. In experiment 7 we use $n = 100$ and $l = 10$. For experiment 8 we use $n = 100$ and $l = 50$. Finally for experiment 9 we use $n = 300$ and $l = 50$. In all three experiments, for every action, after the red or blue region is picked the transition matrix is generated uniformly in the blue region with $p_{12} \in [0.6, 0.7]$ and $p_{21} \in [0.85, 0.95]$ or the red region with $p_{12} \in [0.25, 0.35]$ and $p_{21} \in [0.2, 0.3]$. In Figure 3.14 the results for these three experiments are shown. The performance is once again visualised with a boxplot of the accuracy over 1000 trials.

Similar to Section 3.6.1 there is no significant difference in performance between clustering on the averaged pure process and clustering on the mixed process. Comparing the accuracy spread of Figure 3.14 with that in Figure 3.13 and 3.12, we observe a larger sample variance in the accuracies.This observations comes from the differences between experiments 1 and 4 and experiment 7, the experiments 2 and 5 and 7, and experiments 3 and 6 and 9. This spread can be explained by the fact that the transition matrices are picked in a random fashion. If all matrices are picked from a single region this experiment will show similar performance to the previous figures. However, for the majority of the trials, these matrices will be picked in equal proportions from both regions and therefore the averaged matrix is in the middle of these two regions. This averaged matrix lies in the infeasibility region, which results in a lower accuracy. Likely, this same effect is also the cause of the difference in sample variance between experiments 7 and 8. Because experiment 8 has an increased number of actions the average matrix is likely to be in the middle of the two regions compared to the averaged matrix of experiment 7. Similar to experiments 3 and 6, increasing the number of states in experiment 9 gives less spread in the accuracy and a more consistent accuracy compared to experiment 8.

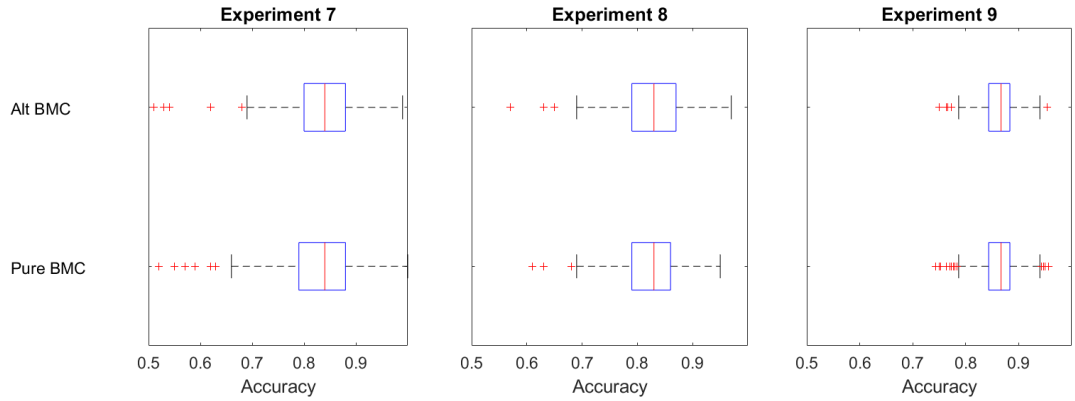


Figure 3.14: Accuracy of the clustering algorithm for a mixed Markov process and a pure markov process. Every boxplot of the accuracy is based on 1000 trials with a trajectory length of $T = n \ln n$.

Examining all experiments, we conclude that for an mixed Markov process and a pure Markov process the CA has identical performance for both the mixed and the pure process. We thereby confirm the theoretical statements in Section 3.5 about the similarities of these two processes from the point of view of the CA.

4

Cluster Markov Decision Processes

In this chapter we investigate the combined approach of clustering on a trajectory and a MDP. We start this chapter with the model formulation we use in Section 4.1. After the model, in Section 4.2 we will introduce the algorithm. After the model and algorithm we describe the extension to the BMC simulator in Section 4.3. In Section 4.4 we will give some examples and analysis of our clustering algorithm. Finally, in Section 4.5 we put some concluding remarks on our algorithm.

4.1 | BMC–MDP model formulation

We start by introducing the Block Markov Chain Markov Decision Process (BMC–MDP) model. Our objective is to incorporate a cluster structure into a MDP. The framework is inspired from the Rich-Observation Markov Decision Process (ROMDP) framework by [16]. With specific assumptions on the shape of the BMC, the ROMDP framework can be altered to fit the BMC–MDP structure and vice versa.

Similar to what we did for MDPs in Section 2.4.2, we will now give a graphical description for the BMC–MDP. This description is shown in Figure 4.1. Our objective now is to incorporate the BMC transition matrix into the description. We assume that the states in blocks (clusters) in the BMC share, besides similar transition probabilities, an identical reward distribution. This results in a framework where the state S_t at time step t is part of a cluster. The cluster at time step t will be denoted by C_t, i.e., cluster $C_t = \sigma(S_t)$. The agent selects an action A_t and then the agent receives a reward R_t, this reward depends on A_t and C_t. Afterwards, the environment goes to the next state S_{t+1} depending on the action A_t and the previous state S_t.

This BMC–MDP can be described by the tuple $M = \langle \mathcal{C}, \mathcal{S}, \mathcal{A}, R^M, P^M \rangle$. Compared to the regular MDP tuple we have added the cluster set $\mathcal{C}$. Recall that $\mathcal{S}$ and $\mathcal{A}$ denote the sets of states and actions, respectively. The elements of these sets are $C_t \in \mathcal{C} = \{1, \dots, K\}$, $S_t \in \mathcal{S} = \{1, \dots, n\}$, and $A_t \in \mathcal{A} = \{1, \dots, l\}$. Here the number of clusters is assumed to be less than the number of states, i.e., $K \leq n$. As indicated in the graphical description, we consider the rewards to depend on the current cluster and action. In the reward we allow for multiple modelling parameters that constitute the reward structure. The reward function R could be of any form as long as $R^M : \mathcal{C} \times \mathcal{A} \to \mathbb{R}$. Here every cluster–action pair can contain multiple variables depending on the chosen reward structure. For example, this reward could be a single variable for the probability in a Bernoulli distributed reward or the mean and variance if the reward distribution is Gaussian. The dynamics of this MDP are described by the transition probabilities between states given the action $P^M_{i',i,j} \triangleq \mathbb{P}(S_{t+1} = i' \mid S_t = i, A_t = j)$, given that $\sum_{i' \in \mathcal{S}} P_{i',i,j} = 1$. Here $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$. These transition dynamics are equivalent to the dynamics of a regular MDP.

We can map the BMC–MDP to an observable MDP $M^{obs} = \langle \mathcal{S}, \mathcal{A}, R', P' \rangle$. The observable MDP is a translation to a regular MDP tuple on the state set that is observable to the agent. This transition is done in the following way: given that s belongs to cluster $\sigma(s)$ the reward structure satisfies $R'(s, a) = R^M(\sigma(s), a)$. This mapping follows from the observation that it is possible to map the cluster reward function to a state reward function if the cluster of the state is known. Hence, the reward for the state–action pair (s, a) equals the hidden underlying cluster–action pair reward for that state $(\sigma(s), a)$. The transition matrix $P' = P^M$ as the transition tensor is already state dependent in the BMC–MDP.

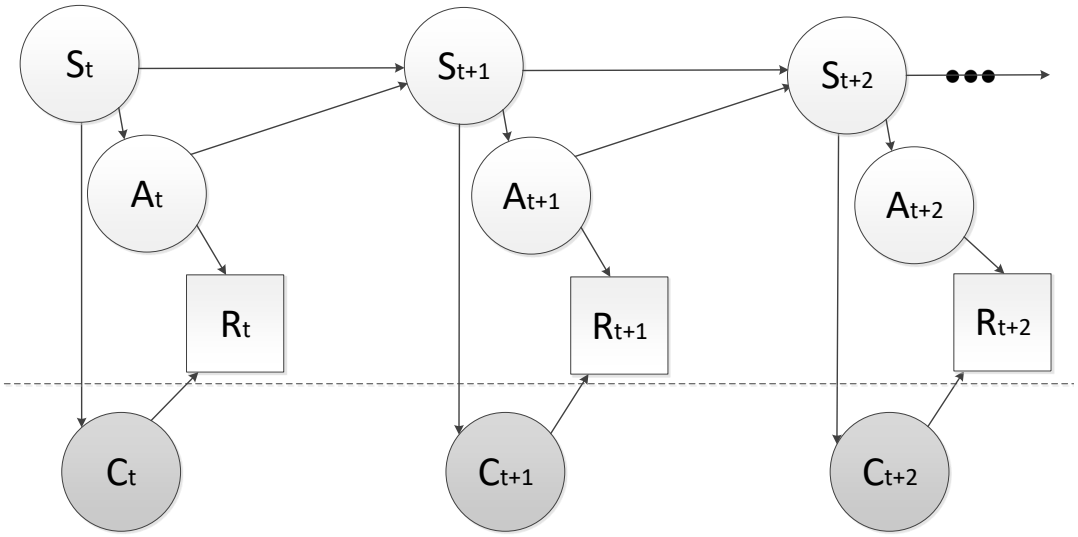


Figure 4.1: Graphical description of a BMC–MDP.

4.1.1 | ROMDP

We will now describe the ROMDP framework [16] and afterwards we will describe how we can alter the BMC–MDP formulation, including the requirements, to fit in the ROMDP formulation. We do this so that we can also identify the differences between the two model and motivate why this new model is introduced.

We describe the ROMDP model using a tuple $M = \langle \mathcal{C}, \mathcal{S}, \mathcal{A}, R, T, O \rangle$. In Figure 4.2 a graphical description of the ROMDP is shown. The major difference is that in this model the dynamics are driven by the clusters. The cluster is not observable for the agent, only the state $S_t \in \mathcal{S}$ is. Similar to the previous MDP models $\mathcal{A}$ is the action set. At time step t we are in cluster $C_t \in \mathcal{C} = \{1, \dots, K\}$. The next cluster C_{t+1} is determined according to the distribution described by the transition tensor T. The transition probabilities between different clusters are given by $T_{k',k,j} \triangleq \mathbb{P}(C_{t+1} = k' \mid C_t = k, A_t = j)$. Here, $T : \mathcal{C} \times \mathcal{C} \times \mathcal{A} \to [0, 1]$. Because this tensor contains transition probabilities, we must have $\sum_{k' \in \mathcal{C}} T_{(k',k,j} = 1$. In this model the agent only observes the current state S_t and not the underlying cluster. To determine the current state from the current cluster we use the observation matrix O. The transition to a given state given the cluster follows by $O_{i,k} \triangleq \mathbb{P}(S_t = i \mid C_t = k)$. Hence the observation matrix maps $O : \mathcal{C} \times \mathcal{S} \to [0, 1]$, and again we must have $\sum_{i \in \mathcal{S}} P(S_t = i \mid C_t = k) = 1$. Note that each observation S_t can only be generated by one hidden cluster, and thus $\mathcal{C}$ can be seen as a non-overlapping clustering of the states. The rewards are determined by the current cluster and action. This reward matrix is described by R in the tuple.

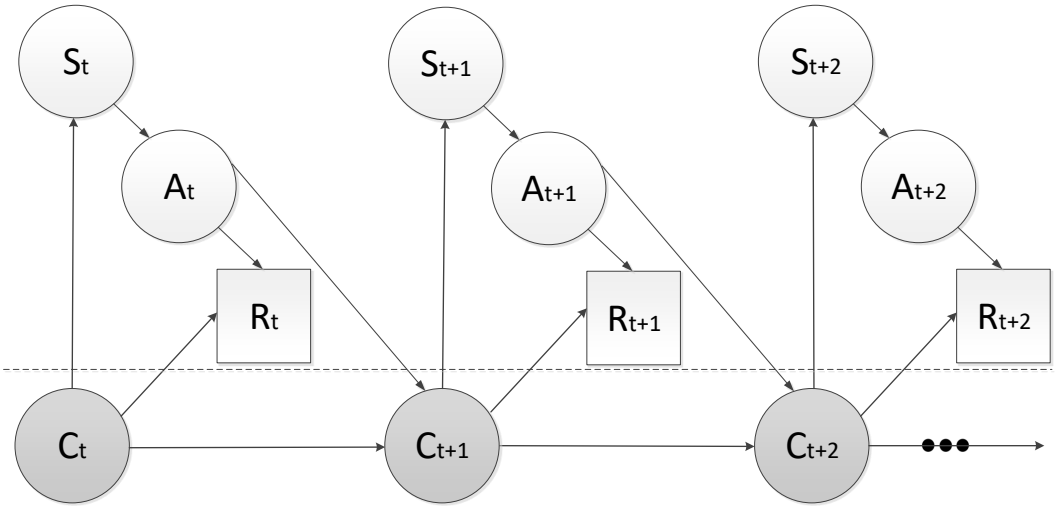


Figure 4.2: Graphical description of a ROMDP.

The ROMDP model can also be mapped to an observable MDP $M^{obs} = (\mathcal{S}, \mathcal{A}, R', P')$ [16]. The expected reward satisfies $R'(s,a) = R(\sigma(s),a)$ given that s belongs to cluster $\sigma(s)$. This is similar to the mapping in the BMC–MDP model. The transition probabilities to the next state i', given the current state i and action j, can be obtained as $P'_{i',i,j} = \mathbb{P}(S_{t+1} = i' \mid S_t = i, A_t = j) = \mathbb{P}(S_{t+1} = i' \mid C_{t+1} = \sigma(i'))\mathbb{P}(C_{t+1} = \sigma(i') \mid C_t = \sigma(i), A_t = j) = O_{i',\sigma(i')}T_{\sigma(i'),\sigma(i),j}$. Hence P' is a multiplication of the cluster to cluster probabilities and the cluster to state probability. This equation follows from the Markov property, the assumption that the clusters are non overlapping, and the law of total probability for conditional probabilities.

4.1.2 | Difference between the ROMDP and the BMC–MDP models

The difference between the ROMDP and the BMC–MDP model is that the environment dynamics are determined by the clusters in the ROMDP model and by the states in the BMC–MDP model. In the ROMDP model the cluster is mapped to a state by the observation matrix. This observation matrix is independent of the previous cluster or state and this results in a constant mapping. In this setting there is no dependency on the previous state. However, in a BMC we do not allow for self jumps. In the ROMDP model it is not possible to model this dependency on the previous state. If the block structure of the BMC contains identical transition probabilities to every state within that cluster, independent of the previous state, then we can rewrite the BMC–MDP model directly to the ROMDP model. This translation is possible because the dependency on the current state disappears and the dynamics can then be described by the current cluster and a mapping from clusters to states.

The BMC–MDP framework is capable of modelling softer cluster structures. The dependency on states opens up more versatility for different kernels that do not have a identical mapping from a cluster to all states within that cluster. For example, consider a cluster structure in which the states in each cluster have identical reward distributions, but in which the state transitions are slightly different for all states in a cluster. Another possibility is a softer BMC, perhaps a MDP in which the sum of all transition probabilities to states within a cluster sum to the intercluster transition probability, i.e., $\sum_{i' \in \mathcal{C}_k} P^M_{i',i,j} = p^{[j]}_{\sigma(i),\sigma(i')}$ for any $k \in \mathcal{C}$ and $j \in \mathcal{A}$. Here $p^{[j]}_{\sigma(i),\sigma(i')}$ is the intercluster probability from cluster $\sigma(i)$ to $\sigma(i')$ given the action j. However, all the transition probabilities to a state in this cluster are not necessary equal, i.e., $P^M_{i',i,j}$ does not necessary equal $P^M_{i'_2,i,j}$ for $i', i'_2 \in \mathcal{C}_k$ for all $k \in \mathcal{C}$. This kind of setting is able to model the random removal of edges within the state space which results in a lower number of states that are reachable from a particular state, but if the transition matrix remains approximately low rank there is still a single communicating class and a clustering methodology could be applied if the intercluster probabilities are constant for the clusters in the transition matrix.

4.2 | Algorithm structure

Now that we have defined the BMC–MDP we will introduce a algorithm structure to find an optimal policy for this BMC–MDP. We will assume that for every state in a cluster the same action is optimal and therefore map the BMC–MDP to a MDP with the cluster space as its state space. In this MDP we estimate the intercluster transition matrix and reward matrix.

We call our algorithm on the BMC–MDP *Cluster Posterior Sampling for Reinforcement Learning (C-PSRL)*. In Algorithm 6 the pseudo code for C-PSRL can be found. The algorithm consists of three phases that we will describe to get a general feeling of how the algorithm works. In the first phase, the actions are selected uniformly at random. The states, actions, and rewards are all stored in memory until we have seen T_c time steps; the clustering length. In the second phase we run the CA to cluster all states using the state trajectory we encountered during the first phase. Finally, in the third phase, we use this cluster assignment to aggregate the data collected in the first phase and solve the MDP using the obtained cluster structure using a RL approach called PSRL. Recall that PSRL is described in Section 2.5.2. We will now describe these three phases in more detail, as well as the design choices that have been made and the parameters that are input to the C-PSRL algorithm. The pseudo-code for C-PSRL can be found in Algorithm 6.

The algorithm starts with an initial phase. The goal of this phase is to pick actions for the MDP and to end with a state trajectory that can be used for clustering with the CA. In Section 3.5 and Section 3.6 we have seen that if each action is picked randomly with some fixed probability and independent of the past, the trajectory generated by such a mixed Markov chain can be clustered using the CA. We will therefore choose to select each action randomly and independently with the same probability in the initial phase. The input that is required for the C-PSRL in this phase is the length of the initial phase T_c and the starting state S_0. For an accurate recovery of the states the clustering length T_c has to be at least $n \log n$, and depending on the information quantity of the problem the clustering length should be picked carefully. The initial state S_0 is used as the initialization of the history $\mathcal{H}_0$. The history stores the action A_{t-1}, state S_t and reward R_{t-1} for every time step $1, \dots, T_c - 1$. At the time horizon T_c, the history $\mathcal{H}_{T_c}$ thus contains the state, action, and reward seen, taken, and obtained at every $0 \leq t \leq T_c - 1$ and the current state S_{T_c}.

The second phase of the C-PSRL is the clustering phase. The CA is ran on the state trajectory $\{S_0, \dots, S_{T_c}\}$ taken from the history $\mathcal{H}_{T_c}$. The output of the CA is the cluster assignment $\hat{\mathcal{C}}$. Because we use the CA the parameters of the CA are required as input to the C-PSRL. These parameters are the number of clusters K and the number of states n. Using the obtained cluster assignment $\mathcal{C}$ the data in the history $\mathcal{H}_{T_c}$ is aggregated and estimates of the transition matrix and reward structure are made for every cluster, action pair. These estimates will be used in conjunction with the posterior distribution to obtain the sample MDP $\hat{M}$. We derive the policy μ by finding the optimal policy for this $\hat{M}$, this policy μ will be used to select actions in the third phase.

The third phase starts by executing the policy μ found in the second phase. The policy is re-evaluated whenever an evaluation function $f_c(\mathcal{H}_t) = 1$. This evaluation function describes the condition when the algorithm will sample a new MDP $\hat{M}$. The evaluation function could be with fixed time steps or a history based function, in Section 4.2.3 we will describe this function in more detail. We find the optimal policy for this newly sampled $\hat{M}$ using backward induction to get the next policy μ. This policy μ will be used until the evaluation function equals one again. When this function equals one again the process is repeated, and this continues until we reach a finite time horizon T_{hor}.

In the overall design of the C-PSRL algorithm we have made various design choices that will be discussed now, in Sections 4.2.1 – 4.2.4.

4.2.1 | Single clustering moment

We start by examining the case of having a single clustering moment. The alternative would be to cluster at multiple time instances. However, the focus of this thesis is to explore whether a combined approach of clustering, using the CA, and solving on the latent MDP is feasible. One of the potential ways to do this is by looking at the regret bounds. If we use a single clustering moment this could potentially make the analysis of this regret feasible as it does not involve multiple clustering moments with potential changes in the cluster assignment. This is why, for this feasibility study, we opt for a single clustering moment. The C-PSRL consists of a single clustering moment, namely after T_c time steps. Before the CA, the random actions imply that the regret is linear up to the clustering time T_c. This linear

Algorithm 6: Pseudo-code for C-PSRL

Input: Initial state S_0, number of clusters K, number of states n, time horizon T_{hor}, clustering length T_c, evaluation function f_c and posterior distribution Φ

$\mathcal{H}_0 = \{s_0\}$
while $t = 0, \dots, T_c - 1$ **do**
  select A_t uniformly at random
  receive reward R_t, observe new state S_{t+1}
  update history: $\mathcal{H}_{t+1} = \mathcal{H}_t \cup \{A_t, R_t, S_{t+1}\}$
end
while $t = T_c$ **do**
  select A_t uniformly at random
  receive reward R_t, observe new state S_{t+1}
  update history: $\mathcal{H}_{t+1} = \mathcal{H}_t \cup \{A_t, R_t, S_{t+1}\}$
  run Clustering Algorithm on state trajectory in $\mathcal{H}_t$ and obtain $\hat{C}$
  sample MDP $\hat{M}$ using $\mathcal{H}_{t+1}$ and $\hat{C}$ using the posterior distribution $\Phi(\cdot \mid \mathcal{H}_{t+1})$
  compute policy μ
end
while $t = T_c + 1, \dots, T_{hor}$ **do**
  if $f_c(\mathcal{H}_t) = 1$ **then**
    sample MDP $\hat{M}$ using $\mathcal{H}_t$ and $\hat{C}$ using the posterior distribution $\Phi(\cdot \mid \mathcal{H}_t)$
    compute policy μ
  end
  execute policy μ
  $\mathcal{H}_{t+1} = \mathcal{H}_t \cup \{a_t, r_t, s_{t+1}\}$
end

regret is determined by the multiplication of the probability of every action multiplied by the immediate regret of that action. If the regret of the algorithm after the clustering step can be bounded, this directly yields a bound on the regret of C-PSRL. Nevertheless, an algorithm that contains multiple clustering moments could potentially yield better performance in terms of regret or the number of optimally picked actions.

4.2.2 | PSRL after clustering

In C-PSRL we first choose actions randomly, then once the states are clustered, PSRL is applied to the lower dimensional problem. However, for the last stage we could have taken any RL algorithm instead of PSRL. In this section we motivate why we chose PSRL over other RL methods.

The majority of papers that combine a clustering method with a MDP opt for a Optimism in the Face of Uncertainty (OFU) based RL algorithm, see for example [15] and [16]. The OFU algorithms, see Section 2.5.1, are provably efficient, but can perform poorly in practice. Rescaling of the confidence sets has been suggested to obtain better empirical performance, e.g. see [41]. A study on the performance of OFU–RL algorithms and PSRL on a finite horizon episodic MDP has been conducted by [42]. This study empirically shows that PSRL outperforms algorithms driven by optimism and they derive a regret bound for PSRL in a finite-horizon episodic MDP. This paper also gives insight in the shortcomings of OFU–RL in terms of the computational tractability. This problem even exists in the bandit problem, the confidence set of UCB has to be rectangular to be computational tractable instead of the efficient ellipsoidal confidence set, as efficient optimization over this ellipsoidal confidence set can be NP-hard. Whereas Thompson Sampling (TS) remains computational tractable as it only generates a single problem instance. It is likely that similar problems cannot be overcome in the MDP problem.

At the start of this project we gained insight into RL by empirically looking at the performance of various bandit algorithms. In various simulations TS performed better than UCB. The only difficulty was to select a good prior distribution as opposed to a properly scaling confidence bound. When we analysed the extension to the MDP problem, and considered the different algorithms that were available, the natural extension of TS; PSRL, had the same difficulty of selecting a prior compared to

the selecting an appropriate scaling confidence sets in the OFU methods. This difference is highlighted because in a MDP sampling from a prior distribution stays the same, there is just the extension to a transition and reward prior. However, in OFU based methods you end up with a confidence set over two unknown variables over which you have to optimise.

Using a prior distribution is an elegant yet simple solution and if we can formulate a proper prior distribution it is a statistical efficient estimator with a low computational requirement compared to tuning confidence bounds. On top of that, the problem of finding proper scaling of these bounds still remains. Nevertheless, when we combined the study of [42] with the earlier bandit observations, we opted for using PSRL as a first study.

4.2.3 | The evaluation function f_c

Next we will determine that the evaluation function should have increasing length intervals between evaluation moments while the algorithm is running. The evaluation function f_c defines the moment when the algorithm re-optimizes and determines a new policy by using the posterior distribution and the history. This is essentially a function to determine how often you want to re-evaluate the current policy and adjust it. We use an evaluation function because the majority of the computational load of our algorithm in the third phase comes from re-adjusting the policy. Note that when we have the information of a large number of time steps the extra information that one extra time step provides will most likely not be enough to change the policy. This warrants our choice not to re-evaluate the policy at every time step. By making this choice the computational load of the algorithm decreases. However, at the start of the agents' interaction with the environment the extra information of a time step is significantly higher than the information gained when nearing the time horizon. Hence, the evaluation function should be of increasing length between re-evaluation moments. The necessary increases of re-evaluation intervals rises the question what the lengths of these intervals should be. There are two potential solutions: we can fix predetermined intervals, or we can choose intervals based on the data seen by the algorithm. With fixed intervals the re-evaluation is independent of the actions. If that is the aim of your algorithm an evaluation function that describes fixed intervals is a good solution.

However, for a general application an interval based on the history is likely superior. This statement is motivated by the fact that these intervals should contain enough extra information for the re-evaluation to make sense and not solely based on the time that passed since the last evaluation moment. In our implementation we therefore opt to pick a history based evaluation function. At each evaluation moment we store the number of times that every estimated cluster–action pair is seen. We store this in the variable $N^h_{c,a}$. Here h denotes the number of evaluation moments that the agent encountered. When one of these estimated cluster–action pairs is seen a multiple of c_e times, the re-evaluation is executed and the number of cluster-action pairs of evaluation moment $h+1$ is stored again. This history based evaluation function is defined as

$$f_c \triangleq \begin{cases} 1 & \text{if } N^{h+1}_{c,a} \succ c_e N^h_{c,a} \text{ for any } c \in \hat{\mathcal{C}}, a \in \mathcal{A} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Here $\succ$ is used to illustrate that not all entries but only a single entry has to be c_e times larger. For example, suppose we take $c_e = 2$. This implies that we re-evaluate the policy when any cluster–action pair is executed twice as much compared to the last evaluation moment. Note that if we take $c_e = 2$ we have double evaluation epochs similar to the execution framework of [34]. This increase in the number of data samples warrants a re-evaluation.

4.2.4 | The prior distribution

One of the key features of the PSRL algorithm is that prior information about the BMC–MDP structure can be used in the prior distribution Φ. This prior distribution is used to sample the transition probabilities and the reward matrix of a MDP at every re-evaluation moment. For PSRL it is suggested by [42] that a uniform Dirichlet prior for transitions and a $N(0,1)$ prior for rewards are a simple way to encode having little prior knowledge. The Dirichlet distribution is a multivariate Beta distribution. Note that the array of observed transitions to every state given the current state action pair follows a multinomial distribution. Nevertheless in PSRL a Dirichlet prior is used. We think that this choice is made because both distributions have the same mean and variance. The benefit of the Dirichlet distribution is that the output is a transition vector to all possible states and that it uses the observed transitions as input.

For the reward a normal distributed prior is chosen as a general model. The estimated mean will then converge to the real mean and by using a Gaussian distribution we model how certain we are of our estimate through the estimated variance. In research on MDP algorithms, the assumption often is made that the transition dynamics are harder to grasp than the reward dynamics. The reward is easier to learn because the reward matrix is independent of the next state, the transition tensor has $|\mathcal{S}||\mathcal{A}||\mathcal{S}|$ entries and the reward matrix has $|\mathcal{S}||\mathcal{A}|$ entries. Therefore, in MDP algorithm research the focus often lies on what speed the algorithm can learn the transition matrix.

In our combined approach we grasp most of the transition dynamics of the environment through clustering because the estimated transition matrix is used in the spectral step of the CA. If we have an adequate number of samples to estimate these matrices for every state–action pair it is likely that we also have enough information to estimate the reward for that state–action pair. In our cluster based approach this effect is moreover enhanced because we aggregate data of all the states in a cluster. This gives us the incentive to use the estimated mean as a prior for the reward structure of the MDP for every cluster–action pair. Nevertheless, if the reward distribution is known a priori we can utilise this information by selecting the proper prior distribution. For example, if the rewards are Bernoulli distributed we can keep track of all the fails and successes of the reward and use a Beta distribution to sample the reward, similar to TS. Compared to using the estimated mean this method prefers exploration as the uncertainty of all estimates is taken into account. The effect of these kind of prior reward distributions is investigated later in Section 4.4.2.

4.3 | BMC–MDP simulator

To simulate the BMC–MDP and implement our C-PSRL we have extended the BMC simulator described in Section 3.2. The two classes, the environment class and the agent class, have remained the same. Both classes have received extra functions. This section describes our additions to these two classes, as well as additions to the utility functions.

4.3.1 | Environment Class

To the environment class of the BMC simulator we add a *Calculate Q-values* function. The original functions, adopted from the BMC simulator, are extended to handle multiple transition matrices for the trajectory generated. Besides this extended dynamic the reward structure is incorporated.

- The initialise function takes as input a transition tensor. Besides the transition tensor, it requires a reward matrix. This reward matrix contains all parameters needed to describe the reward distribution for every cluster action pair. Providing an initial state is still possible as it is an optional input.
- The advance function now requires an input, which stands in contrast to the advance function in the BMC simulator. Specifically, this input is an action. Based on this action and the current state, a reward is output using the reward distribution given by the initialisation. Besides a reward, a transition matrix is selected based on the action taken, and the next state is then determined.
- The reset function remains the same.
- The Calculate Q-values function is a new function that is used to determine the optimal policy and the regret of an agent. This function uses backward induction, Algorithm 1, to determine the value function at every time step. The input required for this function is therefore the time horizon T_{hor}. The output is the value function for every cluster–action pair for every time step $0 \leq t \leq T_{hor}$. The difference in the value function between the optimal action and the selected action is used to calculate the regret.

Besides extending the environment class the supporting functions were also extended to generate input parameters for the BMC–MDP setting.

4.3.2 | Agent Class

The Agent class is extended with a third function, the *select an action* function. The way an agent operates is programmed into each agent. However, the agent always interacts with the environment using these three function so that using every agent is identical.

- The initialise function creates the agent. To initialise the agent this function requires: the clustering length, the final time horizon, the number of clusters, the number of states, and the number of times the CIA has to be run in the second step of the CA as input. Besides these inputs, there

are optional arguments for the initialise function. These optional arguments determine whether the state and action trajectories have to be stored in memory.
- The update function takes as inputs the recent observations: the selected action, the received reward, and the new state. The history is updated and based on the evaluation function it determines if re-evaluation of the policy is needed. If re-evaluation is required, the new policy is determined.
- The select an action function outputs an action of the agents based on the current policy of the agent and gives this action as output.

4.3.3 | Utility functions
The objectives of the utility functions in our BMC–MDP simulator are identical to the objectives of the utility functions in the BMC simulator. In the BMC–MDP simulator the utility functions include for example, functions to plot the regret using the value function, as well as the action and state trajectories.

4.4 | Examples
4.4.1 | First glance on Cluster PSRL
To get a feeling for the performance of C-PSRL we empirically compare the performance of C-PSRL to a comparable implementation of PSRL on the states. This is not a fair comparison because the C-PSRL uses more information, namely that the states can be described using a lower dimensional number: the clusters. Our experiment gives insight into what utilising this extra information can do. We investigate the performance of this algorithm using the regret. Here the immediate regret for every decision epoch is the difference in the value function between the action taken and the optimal action with the highest value function as seen in (2.11).

To compare both algorithms we take a finite horizon trajectory of length $T_{hor} = 2000$ and we run 10 different trials. These trials are on a MDP in which the reward means and the transition matrices are different for every action. These differences are large so that for every action, for the majority of the clusters, the next state is likely to be in a different cluster for every action. This means that the optimal policy can be learned quickly using standard MDP techniques. For the C-PSRL a clustering time $T_c = 100$ is used. As a first experiment, we provide the real cluster assignment to the algorithm at the clustering time. With the correct cluster assignment as input to this simulation, this experiment will give a first insight into the performance of C-PSRL without the interference of clustering errors. The evaluation function f_c is set to a history based evaluation function, see (4.1), with $c_e = 2$. Hence if any cluster action pair is observed twice as often compared to the previous evaluation point the algorithms re-evaluates the policy. For the state implementation there is no clustering step, so PSRL is ran starting at the first time step on the state using the same prior distribution and evaluation function. The problem consists of $n = 8$ states containing $K = 3$ clusters of sizes $\alpha n = (3, 3, 2)$. The number of states is initially picked low in this example to illustrate a minimal situation where clustering can potentially be a positive factor in the performance of a RL algorithm on this MDP. The larger the number of states is compared to the number of clusters, the larger the influence of clustering will become. The number of actions is $l = 3$ and at T_c the correct clusters are given to the C-PSRL. The rewards are Bernoulli distributed with $R = (0.8, 0.7, 0.65; 0.7, 0.8, 0.75; 0.6, 0.6, 0.75)$. The intercluster transition matrix $P^{[l]}$ for all actions $l \in \{1, 2, 3\}$ are

$$P^{[1]} = \begin{bmatrix} 0.0450 & 0.0350 & 0.9200 \\ 0.0125 & 0.0900 & 0.8975 \\ 0.0400 & 0.9000 & 0.0600 \end{bmatrix}, P^{[2]} = \begin{bmatrix} 0.0350 & 0.9200 & 0.0450 \\ 0.8975 & 0.0125 & 0.0900 \\ 0.0400 & 0.9000 & 0.0600 \end{bmatrix}, P^{[3]} = \begin{bmatrix} 0.9200 & 0.0350 & 0.0450 \\ 0.0125 & 0.8975 & 0.0900 \\ 0.9000 & 0.0600 & 0.0400 \end{bmatrix}. \tag{4.2}$$

In Figure 4.3 the results of this simulation are shown. Notice from the blue solid lines that at the clustering length T_c the C-PSRL estimates the reward and transition matrix sufficiently to find the optimal policy in the majority of the trials. In this example, at the time horizon, all trials using C-PSRL found the optimal policy. However, in this case the cluster sizes are small, which warrants the question whether picking a random action for the cluster length is beneficial over optimizing on the larger number of states. In Figure 4.3 you can see that this is not the case for all simulations. This indicates that when the number of clusters is relatively close to the number of states it would likely not be beneficial to use a cluster based RL algorithm over a RL algorithm on the states.

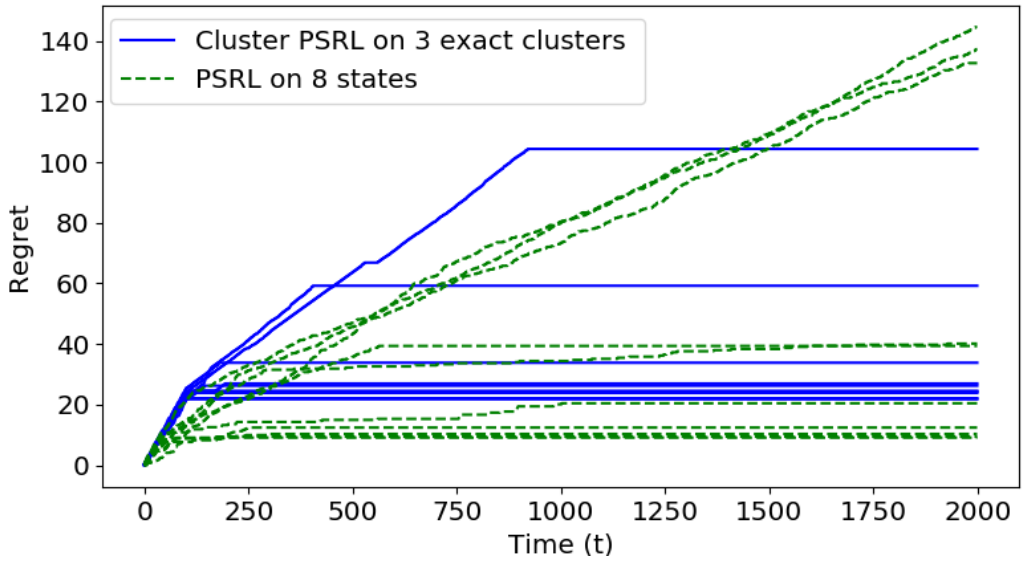


Figure 4.3: The regret for 10 trials of C-PSRL and PSRL on a MDP with $n = 8$ states, $l = 3$ actions, $K = 3$ clusters and cluster sizes of $(3, 3, 2)$. The clustering time $T_c = 100$ and the total finite horizon of this simulation is $T_{hor} = 2000$ time steps.

To show the influence of a larger number of states compared to the number of clusters, we will now increase the number of states. This shows the strength of our clustering methodology. An otherwise identical simulation is ran with the number of states increased to eighty instead of eight with cluster sizes $\alpha n = (30, 30, 20)$. We increase the clustering time to 1250. The clustering length is around $3n \ln n$. In this situation the correct cluster assignment is still given to the algorithm at T_c. Besides this experiment in which we give the exact cluster assignment, we will now also show the results of an experiment in which the clustering is done by the CA at the clustering length T_c. To indicate the difficulty of clustering in this example we give the average intercluster transition matrix

$$P_{avg} \approx \begin{bmatrix} 0.3333 & 0.3300 & 0.3367 \\ 0.3075 & 0.3333 & 0.3592 \\ 0.3267 & 0.6200 & 0.0533 \end{bmatrix}. \tag{4.3}$$

This averaged matrix is computed by multiplying the intercluster transition matrices, seen in (4.2), with a probability of $1/3$ for every matrix. From the intercluster transition matrices in (4.2) you can observe that the first two clusters have different dynamics for the different actions. But the averaged transition matrix in (4.3) shows a matrix with little variation between the transition probabilities of clusters one and two. Therefore, even though the behaviour of the first two clusters are not similar, the entries of the averaged matrix are similar. We expect that, because the averaged intercluster transition matrix of cluster three is different compared to the averaged transition matrix of cluster one and two, the CA is likely capable to cluster the states of cluster three correctly. We can thus expect that it will be difficult for the CA to cluster the states of the first two clusters.

In Figure 4.4 the results of these two experiments are shown besides the results of the PSRL algorithm applied on all eighty states. From Figure 4.4 you can observe that finding the optimal actions for the states using the PSRL algorithm yields a lower regret at the time horizon T_{hor} than the cluster based algorithms with this specific clustering length.

Next we look at C-PSRL's regret in Figure 4.4. The dotted red line indicates that the optimal policy is found for all clusters at T_c when the real cluster assignment is given to the algorithm. This is observed from the horizontal line after the clustering length for all trials. When we compare the dotted red line with the solid blue line we can conclude that the CA is not able to consistently cluster the states correctly with this clustering length. There is a decrease in the slope after the clustering length when the CA is used as can be denoted from the blue line. When we compare the two clusters based curves we observe that, when we inject the exact cluster assignment, the regret curve are horizontal after the clustering

length. When we cluster with the CA we see a linear increasing line. We therefore can conclude that at this clustering length the performance of the CA differs and the linear slope is an indication of the accuracy of the cluster assignment. When we operate in the critical regime, the required clustering length

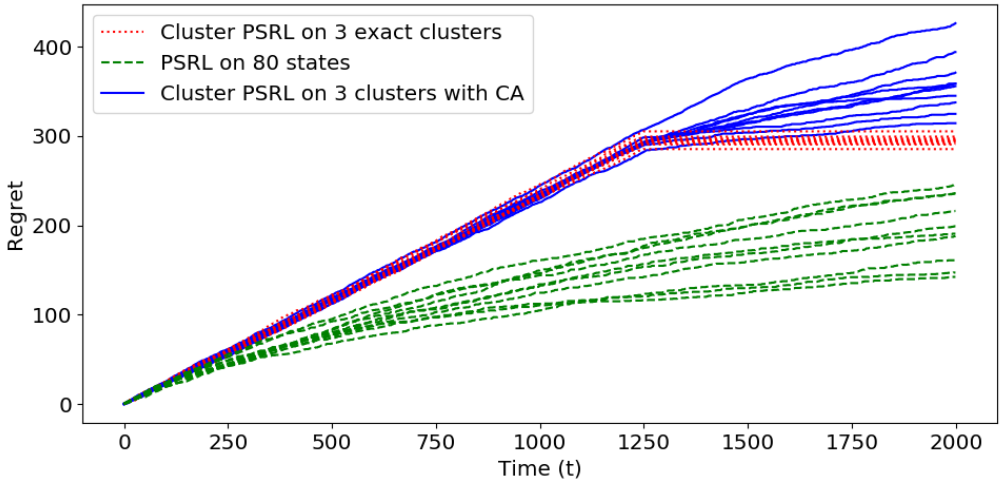


Figure 4.4: The regret for 10 trials of C-PSRL and PSRL on a MDP with $n = 80$ states, $l = 3$ actions, $K = 3$ clusters and cluster sizes of $(30, 30, 20)$. The clustering time $T_c = 1250$ and the total finite horizon of this simulation is $T_{hor} = 2000$ time steps.

for exact detection depends on the information quantity $I(\alpha, p)$ of the averaged intercluster transition matrix. Considering that the feasibility region for the critical regime is defined asymptotic as $n \to \infty$ the clustering length might not be sufficient to cluster the states even if the minimum information quantity condition is met for the asymptotic case. This example illustrates the importance of a correct cluster assignment for the algorithm as all the data of the states in these clusters is aggregated. Therefore we conclude that the cluster assignment and thus the selection of the clustering length T_c, as the algorithm clusters once, is of vital to the performance of the C-PSRL.

4.4.2 | The evaluation function and the prior distribution

Next we are going to investigate the influence of the evaluation function and the prior distribution for the reward that are used to sample the MDP. We investigate the evaluation function because the majority of the computational cost of the C-PSRL, in the third phase, comes from the number of re-evaluations. At these re-evaluation moments the MDP is sampled and the optimal policy is recalculated for this sampled MDP. At the other time points in the third phase the current policy is purely being executed and the history, or the estimates of the prior distribution, is begin updated, and this is computationally not intensive.

Within this section we look at the evaluation function f_c that determines when we re-evaluate the current policy and thus also the start of the next episode. The type of evaluation function we investigate is the history based evaluation function. The condition that has to be met to go into the next episode is $N_{c,a}^{h+1} > c_e N_{c,a}^{h}$, see (4.1). Here $N_{c,a}^{h+1}$ denotes the number of times the estimated cluster action pair c, a is seen by the algorithm at the re-evaluation episode h. The next episode $h + 1$ starts when one of the cluster action pairs is seen c_e number of times. We will investigate the influence of the evaluation constant c_e. We will also look at the influence the prior distribution has on the reward. The prior distribution determines how the MDP is sampled from the history. In Section 4.2.4 we discussed the influence of the prior distribution on the sampled MDP and explained that the choice of prior influences the exploration behaviour of C-PSRL. Within this section we will investigate two types of reward prior: the estimated mean and a Beta prior distribution.

In the previous section we have seen regret curves for various trials in a single figure. However, if the number of trials increases this kind of plots become messy. We will therefore report the regret distribution at a certain time step using a box plot. The box plot reports the median, a box surrounding this median, the whiskers, and statistical outliers. The box reports the spread of quartile one to quartile three, i.e., the middle fifty percent of the data. The whiskers are generated with a maximum length of

1.5 times the width of the box or with a length that contains all sample points. If there are points outside this maximum whisker length they are considered statistical outliers.

First we will show a general regret distribution for a single problem and illustrate the behaviour of this regret curve over time. Besides this we will draw the box plot at the time horizon to illustrate how a box plot reports the distribution at this particular time step. The C-PSRL algorithm uses a history based evaluation function with $c_e = 1.5$ in this experiment. The reward matrix is sampled from a Beta prior distribution, and we choose a time horizon $T_{hor} = 10000$. We optimize for a MDP with transition matrices as in (4.2). The reward is Bernoulli distributed and the means for every cluster–action pair are relative far apart and range from $0.6 - 0.9$. There is at least 0.05 difference between every reward mean. The cluster assignment is injected in this simulation, meaning that the cluster assignment is correct at T_c. For this experiment we run 500 trials.

From Figure 4.5 we can observe by the horizontal line of the third quartile that at least 75 % of the trials converge to the optimal policy. This convergence is observed after around five thousand time steps. Based on Figure 4.5 we conclude that for most trials the regret is stable after a time interval of around four thousand in this case. For reference also the maximum observed regret from these five hundred trials is plotted with a dashed line. The box plot that is obtained at T_{hor} is shown to the right.

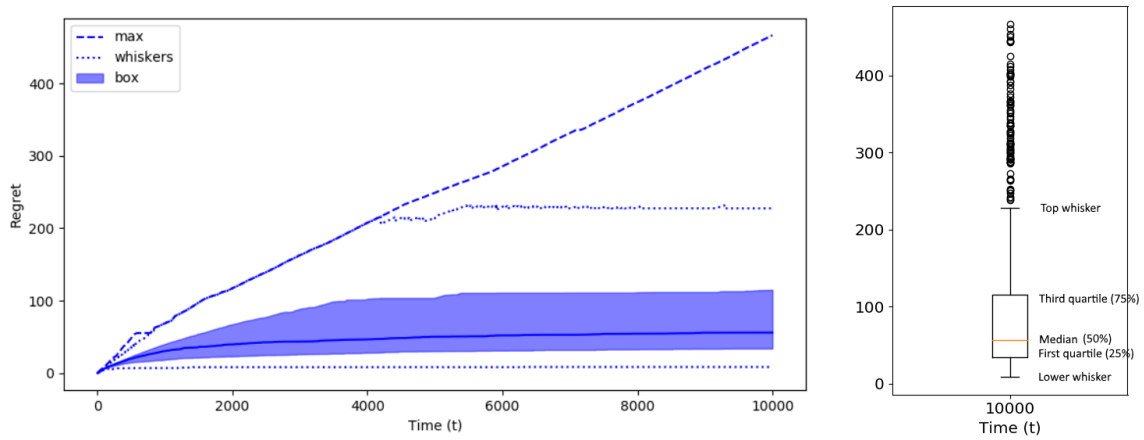


Figure 4.5: Box plot over 500 trials of the regret over a time horizon of 10000 time steps. Here, the evaluation constant is $c_e = 1.5$, and the reward is drawn from a Beta distribution.

The outliers in Figure 4.5 can be explained with the exploration versus exploitation dilemma. The MDP that is used to determine the policy is sampled using a prior distribution. If the estimates of the reward matrix and the transition matrix are not close to the real value, the algorithm keeps believing that a non-optimal action is optimal. This results in a linear regret curve (as seen from the maximum line in this figure). Due to the fact that the estimates depend on a finite number of observations, it is statistically possible that these estimates are far from the true values. However, due to the number of samples, the algorithm is certain about its estimate and starts exploitation of the non-optimal policy that is determined from these inaccurate estimates.

The Spectral Clustering Algorithm clusters by doing a K-means step and a SVD of the transition matrix of the BMC. If we are at sufficient length to cluster the states into the clusters with a high accuracy we can interpret this as the agent having an understanding of the dynamics of the MDP. The challenge for the agent in C-PSRL at this point is understanding the reward structure. Fortunately as we saw in Section 4.2.4, the reward structure is usually the easier structure to learn as there are fewer options to be explored. We will now describe two scenarios that suggest that exploration on the reward is favourable. Both these scenarios result in an agent picking the non optimal action if there is no reward exploration in C-PSRL. This behaviour can be seen in Figure 4.3 in which the green lines are potentially caused by this phenomena, as well as from the maximum line in Figure 4.5.

Assume that we are given a state, and that we set the reward mean in the order of $\sim 1/T_c$. This results in estimates of the reward mean and its variance at the clustering length T_c that are of the same order as the actual mean. This means that the C-PSRL has not grasped the dynamics of the problem at the clustering length T_c, and further exploration is needed to improve the estimate of the reward.

This same phenomena can also be observed in the case that the reward means of two actions are considerably separated.

We will now investigate the first case is a Beta prior distribution. The number of data points will be logged similar to the Thompson Sampling approach for bandits, see Section 2.3.3. We use a Beta distribution on the number of successes and fails for every estimated cluster–action pair and sample the rewards using this distribution. We now also investigate the case in which we directly map the estimated mean to the sampled MDP reward matrix. In this situation we do not take the number of observations of this estimated cluster–action pair into account.

We expect the Beta distribution to be a deciding factor in performance if exploration is needed because the reward means are within a small range of each other for all actions. We use a MDP with transition matrices as in (4.2). The Bernoulli reward mean for all cluster action pairs have a small range and therefore are expected to be difficult to estimate within the clustering length or the horizon length. The reward means are in the range of 0.02 to 0.05 in this experiment. For Bernoulli distributed reward the mean corresponds with the probability of getting a reward of one. The cluster assignment is injected in this simulation, so the cluster assignment is correct at T_c, meaning that we can solely focus on the influence that the prior distribution and the evaluation constant c_e has on the results. In this experiment we run 500 trials. In every trial for all the different $c_e \in \{1.1, 1.25, 1.5, 2, 2.5, 3\}$, the same trajectory up to T_c is used so that the estimates of the transition tensor and the reward matrix for all estimated cluster–action pairs at the first evaluation moment are equivalent. We have $n = 30$ states, $l = 3$ actions with $K = 3$ clusters. The cluster sizes are $(15, 8, 7)$ and we use a clustering length $T_c = 100$ and look at the cumulative regret for a finite horizon $T_{hor} = 500$. We use such a low number of states because we inject the clustering assignment into the CA. This number of states is sufficient to show the general behaviour of both parameters investigated in this section. In Figure 4.7 the box plot of the regret at the time horizon T_{hor} is shown with on the left the box plot of the regret if we use a Beta distribution to sample the reward structure of the MDP and on the right when we use the estimated reward mean. Based on this reward mean we expect to see different behaviour for the C-PSRL because the estimates at the clustering length are probably not accurate enough to distinguish the different means. The box plots of the regret at the horizon of this experiment are shown in Figure 4.6 for both the Beta prior distribution as well as the sampling based on the estimated mean.

From the right plot in Figure 4.6 we can observe that for all c_e the spread of all box plots is in the same range. This can be explained by the fact that if the estimate is correct at the clustering length, the agent will only pick the optimal action and therefore remain a low regret is incurred. On the other hand if the reward mean of the best action is underestimated using this sampling method there is no exploration and this action is never picked with high probability which results in a high regret. Hence you can observe constant performance independent of the evaluation constant, in terms of regret, using this sampling method for all evaluation constants. When we observe the box plots in the left hand plot we can see different shapes compared to the right plot. The median is higher than the estimate version but the boxes have a lower range so the performance of the algorithm when we use a Beta distribution to sample the reward is more consistent. A priori you may expect the minimum regret to be higher because the Beta distribution should enhance exploration and this is the case as the minimum regret value is higher.

The influence of the evaluation constant is clear from the left plot in Figure 4.6. Having a higher evaluation constant means that the algorithm will re-evaluate on fewer occasions compared to when the evaluation constant is small. You can observe the regret value of the outliers to decrease when the evaluation constant goes up. The algorithm performs becomes more consistent when the evaluation constant is high. An action is performed for a longer duration and that results in a larger spread in the box plot, between the minimum and maximum observed regret, when the evaluation constant increases. This is expected because if the optimal actions are taken and the policy is not re-evaluated regularly this yields a low regret and the algorithms can perform like it would when we use sampling using the estimated reward in the right plot.

To test our hypothesis from Section 4.2.4 that states that the estimated mean is a good prior if the reward structure is more distinguishable. We take the parameters of the previous experiment but we change the reward matrix. The Bernoulli distributed rewards means are relative far apart and range from $0.6 - 0.9$ for every cluster action pair and there is at least 0.05 between every reward mean. We plot the results of this experiment in Figure 4.7.

From the right plot in Figure 4.7 we can observe that for all c_e the spread of the regret are in the

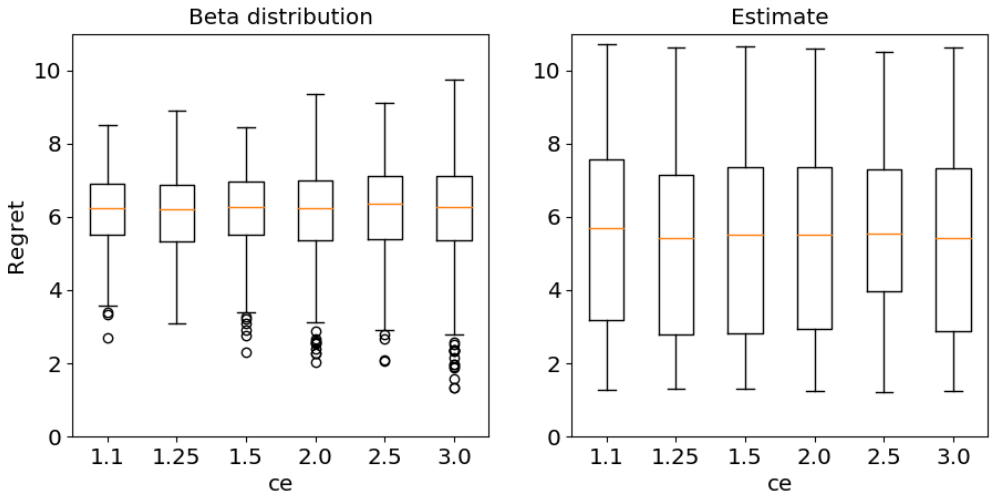


Figure 4.6: Box plots of the regret at the time horizon T_{hor} for various $c_e \in \{1.1, 1.25, 1.5, 2, 2.5, 3\}$ for two different methods to sample the MDP. Left the reward is sampled from a Beta distribution, and on the right the estimated mean is used as the reward for the sampled MDP. The Bernoulli distributed reward means in this simulation are in the order of $0.02 - 0.05$.

same range if we use the estimate mean to sample the MDP. Because the majority of the samples is in the lower range the reward mean at T_c for all cluster action pair are often accurate to distinguish the and this yields the selection of the optimal action at T_c. Because the sampling of the reward is using a greedy policy the policy has a high probability of picking this action for the entire duration. This explains the range of the box plot as the agent continues picking the non optimal or optimal action. But in the majority of the cases the first estimate is close to the real mean as the majority of samples, the box plot range, is in the lower part. If exploration is encouraged, which is done if we use a Beta distribution, you expect that the minimum regret value is higher as the agent tries the various action options for every cluster. If we look at the left plot we can see that this is the case as the minimum regret is slightly higher than the minimum regret in the right plot. The maximum regret on the other hand is lower and this comes from the fact that if the estimate of the MDP are wrong and the action that gives the highest immediate regret is not always picked because we favour exploration. The range of the box plot without outliers is larger and the median is larger than when we use the Beta distribution to sample.

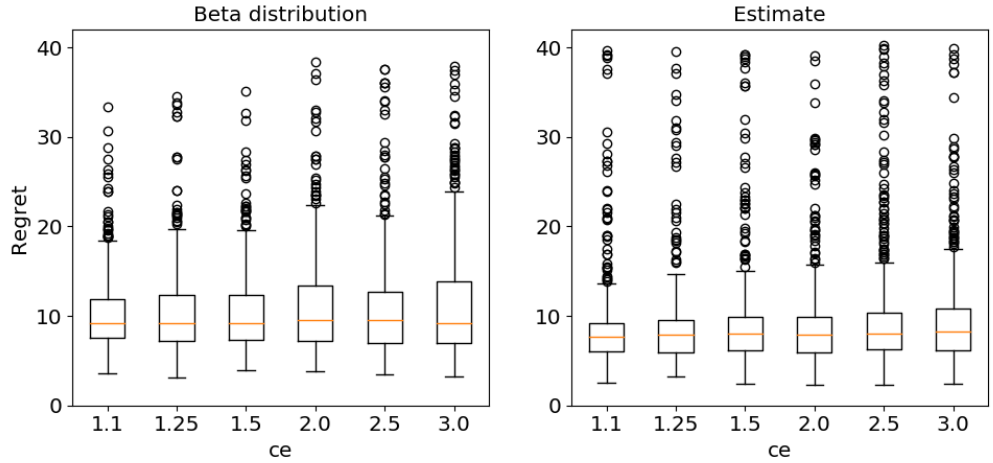


Figure 4.7: Box plot of the regret at the time horizon T_{hor} over 500 trials for various $c_e \in \{1.1, 1.25, 1.5, 2, 2.5, 3\}$ for two different methods to sample the MDP. Left the reward is sampled from a Beta distribution and in the right figure the estimated reward mean is used as the reward mean for the sampled MDP. The reward means in this simulation are in the order of $0.6 - 0.8$.

The evaluation constant c_e main influence is on the computational power required to run the algorithm. The evaluation constant did not influence the median regret found by the algorithm. That

is expected because on average the algorithm the algorithm performs the same, the influence is expected to be on the regret spread of the algorithm. We can conclude from Figure 4.6 and 4.7 that the regret spread that is obtained using a Beta prior distribution is lower than the spread when we use the estimated mean to sample the MDP.

4.4.3 | Comparison of C-PSRL to TS

We are now going to investigate the performance of C-PSRL on a MDP that has a constant transition matrix for every action. This results in a MDP in which the action only determines the reward received and not the dynamics of the system. We can therefore compare our algorithm to a modified version of a bandit algorithm. We do this to see what the difference is between an algorithm that is specifically aimed at optimizing this problem, the bandit algorithm, to an algorithm that should be capable of solving this problem but has additional parameters that it estimates.

We start with an explanation of how we use a bandit algorithm in this situation. If the transition matrix is not dependent on the action, this kind of MDP can be formulated as a clustering problem. Once the clusters are found, we can use a multi armed bandit algorithm on every cluster. This is because it is not possible to influence the dynamics of the MDP so the only thing that can be influenced by the agent is the reward received by picking the appropriate action that yields the highest reward. We will use Thompson Sampling (TS) with a Beta distribution prior. The number of successes and fails is tracked for every estimated cluster–action pair. For C-PSRL we will also have Beta distribution as prior for the reward and we will use a history based evaluation function with $c_e = 1.5$.

We take a MDP with $l = 3$ actions with intercluster transition matrix $P^m = (0.92, 0.045, 0.035; 0.0125, 0.8975, 0.09; 0.04, 0.06, 0.9)$ for all three actions. The reward is Bernoulli distributed with the reward matrix $R^M = (0.02, 0.05, 0.025; 0.1, 0.08, 0.09; 0.09, 0.035, 0.045)$. We have $n = 100$ with cluster sizes $(55, 20, 25)$. We take a clustering length of $T_c = 1000$ and a finite time horizon $T_{hor} = 8000$. For both the TS and C-PSRL we use the CA to cluster the states. We use 500 trials for both algorithms. The initial trajectory for t until T_c is identical for both methods. Hence the initial estimated reward matrix at T_c is identical for both methods.

In Figure 4.8 we can see the results of this experiment. The two left plots show the box plot at the time horizon T_{hor} of both methods. The regret curve is shown in the most right plot. We compare the two box plots and see that the spread in the regret of C-PSRL is larger than the spread of TS on the three different estimated clusters. This behaviour is expected because the in the bandit approach we made assumptions on the dynamics and therefore reduced the complexity of the problem to learning the reward matrix only. In C-PSRL we estimate the transition matrix next to the reward matrix and that results in more parameters that can be estimated poorly. It is however surprising that the mean and median are almost of a similar height for both methods. This means that, for half of the trials, the two algorithms have similar performance. However, it appears that the C-PSRL does poorly in recovering from wrong estimates as the maximum curve in the right plot is linearly increasing. The third quartile (top of the box) and with that the size of the box is notably higher for the C-PSRL.

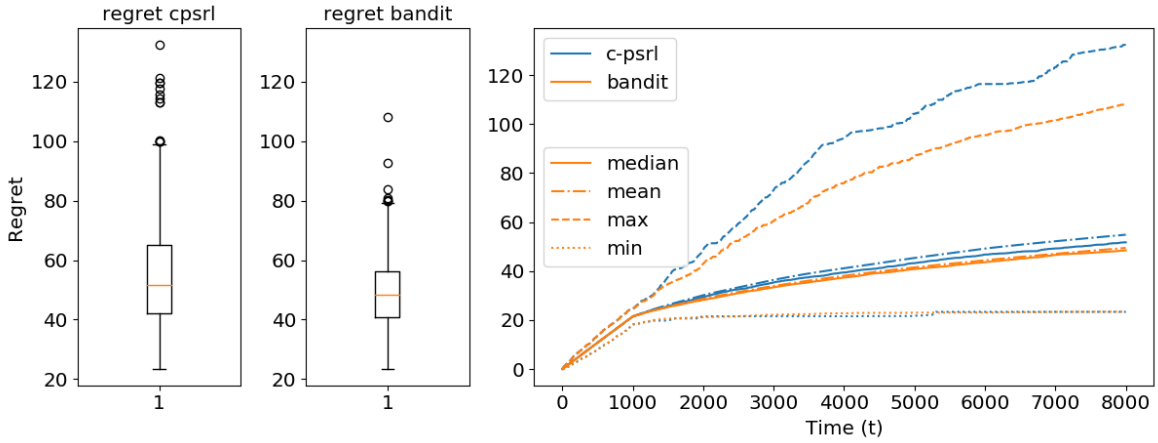


Figure 4.8: Comparison of TS on the estimated clusters to C-PSRL for the case of $K = 3$.

Based on Figure 4.8 we can conclude that a bandit inspired algorithm performs more consistent on this problem. That is to be expected and we hoped, beforehand, that the C-PSRL would perform in a similar manner. If C-PSRL would perform similar, our algorithm would be able to consistently solve lower complexity problems that can be fitted in the framework. We see that for the majority of the trials this is true, but C-PSRL seems to get sometimes stuck in a non-optimal strategy. However, this strategy is still the best strategy to the algorithms knowledge. The algorithm getting stuck in a non-optimal strategy is most likely caused by poor estimates due to a low finite number of samples.

It has to be noted that in this experiment for the bandit algorithm the best action is calculated at every time step. C-PSRL picks the best action according to a policy that is estimated at every re-evaluation moment. If we would use a prior distribution stating that all transition matrices are the same and we use an evaluation function that re-evaluates at every time step, that would make C-PSRL identical to this bandit approach. Technically in the bandit approach we used the correct assumption that there is no dependency on the transition matrix. By picking the proper prior for the transition matrix we could add this extra information into C-PSRL.

4.4.4 | C-PSRL

In the majority of the previous examples we have injected the cluster assignment at the clustering length. We injected this cluster assignment to analyse certain parameters and behaviour of the C-PSRL in the ideal situation that the estimated cluster assignment is correct. We are now going to investigate the performance of C-PSRL when we use the CA and compare the performance to PSRL on the states of the MDP where we do not leverage clustering.

In both Figure 4.3 and Figure 4.4 we have seen that PSRL on states can outperform C-PSRL. Both examples use a Bernoulli reward distribution in which the reward probabilities are far apart. Hence, the difference in these means is easily recognized by the agent. We increase the number of states to $n = 100$ and make the reward similar to each other by choosing $R = (0.02, 0.05, 0.025;$ $0.1, 0.08, 0.09; 0.09, 0.035, 0.045)$. We take a MDP with $l = 3$ actions with intercluster transition matrix $P^m =$ $(0.92, 0.045, 0.035; 0.0125, 0.8975, 0.09; 0.04, 0.06, 0.9)$ for all three actions. The clustering is performed using the CA. We run twenty-two trials for both algorithms and plot the median, the first and third quartile (box of the box plot) along side the maximum and minimum regret we observed in Figure 4.9. You can conclude that for this MDP the PSRL algorithm has difficulty estimating the parameters within this horizon. The regret curves keep increasing linearly and do not show any sign of slowing down. The curves for C-PSRL after the clustering length T_c are decreasing in slope compared to the random actions before we clustered.

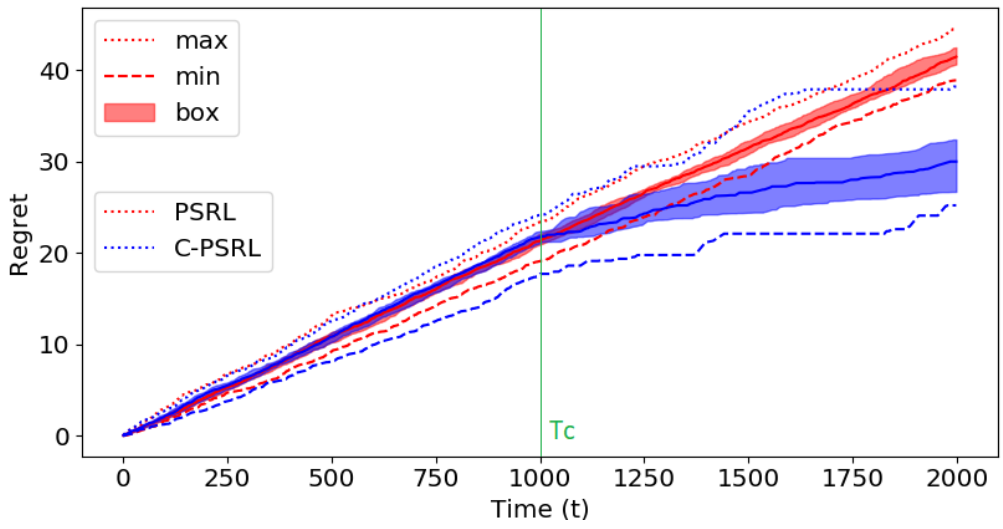


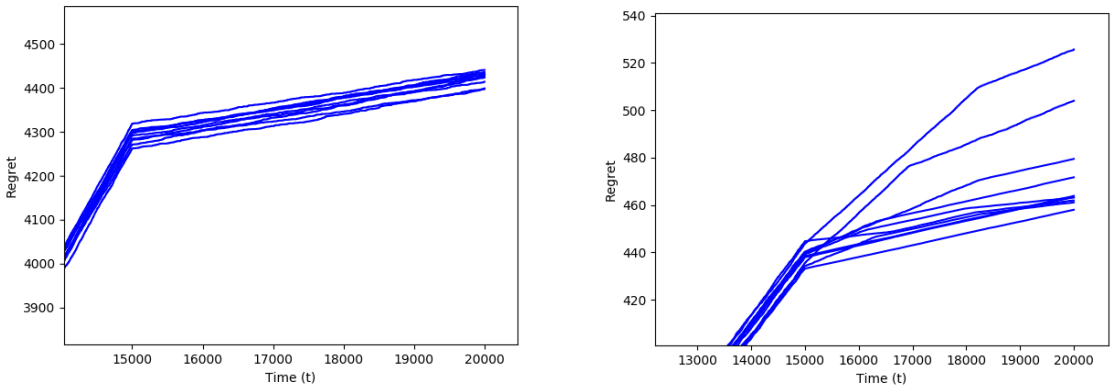
Figure 4.9: Comparison of PSRL and C-PSRL.

Calculating the policy on this number of states is computational heavy. We assume the same number of re-evaluations in both methods, every backward induction step to calculate the policy is an iteration over the size of the action space, then for every action a sum of the entire state space. We do this iteration for all states at every time steps un till to the time horizon. So by using a cluster based approach for a re-evaluation moment at time t we use $(T_{hor} - t)|\mathcal{S}||\mathcal{A}|\mathcal{S}|$ operations and by using a cluster based method the $|\mathcal{S}|$ reduces to $|\mathcal{C}|$. The regret curve in Figure 4.9 makes it clear that a RL algorithm on the total space set cannot compare in performance to a cluster based methodology. Next we will illustrate some examples of C-PSRL on different situations in which we increase the number of states drastically to the region where clustering is of more added value. We have $K = 3$ clusters that contain a total of $n = 500$ states. The cluster sizes are $(200, 125, 175)$. We have $l = 3$ actions and define the intercluster transition matrices for all actions by

$$P^{[1]} = \begin{bmatrix} 0.0450 & 0.9200 & 0.0350 \\ 0.0125 & 0.0900 & 0.8975 \\ 0.0400 & 0.9000 & 0.0600 \end{bmatrix}, P^{[2]} = \begin{bmatrix} 0.0350 & 0.9200 & 0.0450 \\ 0.8975 & 0.0125 & 0.0900 \\ 0.0400 & 0.9000 & 0.0600 \end{bmatrix}, P^{[3]} = \begin{bmatrix} 0.9200 & 0.0350 & 0.0450 \\ 0.0125 & 0.8975 & 0.0900 \\ 0.9000 & 0.0600 & 0.0400 \end{bmatrix}.$$

The results are shown in Figure 4.10. This Figure indicates the simulations for two different reward matrices. We do not show the first time steps until we get close to T_c as the regret curve is just a linear line. Instead we focus on the performance after the clustering step. In Figure 4.10a the results are shown for the reward matrix $R^M = (0.8, 0.3, 0.3; 0.5, 0.8, 0.4; 0.3, 0.3, 0.65)$. When we analyse the slope in Figure 4.10a, it suggests that the clustering performance from the CA is not exact for this trajectory length. From the fact that for all trials the slope is identical, we can conclude that the clustering performance is consistent and fifteen thousand is a sufficient number of samples to distinguish the nine different estimated cluster–action pairs. The linear slope therefore is solely caused by the misclassification of states.

Finally we increase the difficulty in the reward structure by making the Bernoulli reward means closer. Specifically, we set $R^M = (0.02, 0.05, 0.025; 0.1, 0.08, 0.09; 0.09, 0.035, 0.045)$. The results for this otherwise identical simulation are shown in Figure 4.10b. The clustering behaviour is similar to the experiment of Figure 4.10a, because the reward sequence is not taken into account by the CA. By looking at the regret curves after the clustering length we observe that for this trajectory length the C-PSRL is not convinced by the reward structure as we can clearly observe the re-evaluation moments where the slope in the regret curve changes. The majority of the examples end in the same slope as the example in Figure 4.10a, however the algorithm is not confident of the MDP parameters at the clustering length for several trials. We can clearly see the algorithm capability to explore options and getting an improved policy as time progresses.



(a) R^M's means are far spread.

(b) R^M's means are close .

Figure 4.10: C-PSRL on a MDP with $n = 500$ states, $K = 3$ clusters, and $l = 3$ actions.

4.5 | Applicability of C-PSRL

We have seen different experiments in Section 4.4 in which we use C-PSRL on various problems. In this section we will use the insights obtained in Section 4.4 to discuss the applicability of the C-PSRL on

various problems. We will highlight what to consider when applying this algorithm and what is important when you apply it.

In Figure 4.4 we demonstrated the importance of having a correct cluster assignment by looking at the regret curves of both C-PSRL with the exact cluster assignment injected and estimating the clusters using the CA. If the CA is able to exactly cluster and, on top of that, the reward distribution is distinguishable for all cluster–action pairs at that clustering length T_c, we are likely to find the optimal strategy at T_c. This observation follows from both Figure 4.4 and Figure 4.10a.

Just like PSRL, C-PSRL performs better if additional prior information is provided to the agent. The prior distributions can be adjusted to exploit this additional information. Utilising this prior information can make a better suited algorithm that is more consistent in performance as seen in Figure 4.6.

If we are able to estimate the parameters of the problem we can select a clustering length that most probable results in an exact cluster assignment. If we can find these clusters exactly with the CA, we can expect identical performance as learning on this latent MDP directly after the clustering length T_c. In terms of regret, compared to learning directly on this latent MDP, suboptimal actions are only taken prior to the clustering length. After T_c we learn as if we learn directly on this latent MDP; we therefore expect the regret to match the regret of a learning algorithm on this smaller state space MDP if the cluster assignment is exact. Hereby we illustrate the importance of the accuracy of the cluster assignment. It therefore makes sense to apply a clustering methodology on a MDP that is suited to be clustered. If the number of states increases this effect is enhanced; the larger the ratio n/K, the more effective the clustering methodology.

If the number of clusters is not an input but if the algorithm can be relaxed in the sense that it automatically calculates the number of clusters, we could even apply this kind of methodology on MDPs where there is no apparent cluster structure. Depending on the number of clusters the algorithm detects, a suitable choice could be made between the various available learning methodologies.

5

Conclusion

This thesis aim was to investigate and evaluate the combination of the clustering methodology of [40] with Reinforcement Learning (RL). In this Chapter we will describe the research in this thesis that led to the evaluation of this combined approach.

In Chapter 2 we gave an introduction to *Reinforcement Learning* and described the general framework of RL. Afterwards, we described two novel problems: the bandit problem and the MDP. We provide several algorithms that can be used to find a policy in these two problems.

In Chapter 3 we described the problem of clustering using a single trajectory of a block Markov chain. We highlighted the theoretical foundations of clustering in these Markov chains and described the algorithm developed by [40] that is able to cluster on a single trajectory. We implemented a block Markov chain simulator including this algorithm and this simulator allowed us to conduct experiments on this clustering methodology. We also contributed to the revised paper *Clustering in Block Markov Chains* [40] by providing insight in the convergence behaviour of the clustering algorithm and the number of improvement steps that are needed to get an accurate cluster assignment. These insight answered scientific questions that came up in the referee process of [40]. Besides an analysis of the convergence behaviour, we investigated the behaviour of the clustering algorithm in the critical regime. We investigated the feasibility region of the algorithm in the case that there are two clusters. We observed that for a limited number of states the estimated region fits the asymptotically defined feasibility region. By increasing the number of states, we observe that the estimated feasibility region convergences to the asymptotically defined theoretical region. With this observation we conclude that, in the critical regime, the clustering algorithm of [40] meets the theoretical feasibility region. This feasibility region depends on the underlying block Markov chain parameters, if this parameters are known a priori we can reduce the required trajectory length, in a combined approach of clustering with a MDP, for which we can guarantee an accurate recovery of the clusters. The dynamics of a Markov decision process are determined by multiple transition matrices and the clustering algorithm is designed for a pure transition matrix. We compare the statistical properties of a trajectory generated by a mixed Markov chain with the trajectories of a pure Markov chain and we found that, given the action selection protocol, these two processes are similar to each other by the total variation distance. When we look at a single state trajectory, the two processes have the same probability of generating this trajectory. Having investigated the stochastic properties of these two processes, we performed a study on the performance of the clustering algorithm of [40] on a mixed Markov chain. We compared the algorithms performance on a trajectory generated by the mixed Markov process and the pure Markov process. For two clusters with a varying number of states and actions, we conclude that the clustering algorithm has similar performance on both processes. This observation gave us the confirmation that we can use the clustering algorithm of [40] for a state trajectory generated by a MDP.

In Chapter 4 we introduce the *Block Markov Chain Markov Decision Process (BMC–MDP)* model that defines a framework that is combines a general cluster able structure, like a block Markov chain, with a MDP. We gave benefits of this model compared to existing models and highlighted practical extensions in general cluster based transition dynamics that can be described by this general model. We introduced the Cluster Posterior Sampling for Reinforcement Learning (C-PSRL) algorithm which was able to give us insight in the properties of a combined cluster based Markov decision process

approach. We restricted our attention to a single clustering moment and took PSRL as the algorithm we implement in this combined approach. We performed simulations of C-PSRL on Bernoulli reward based MDPs and observe that the cluster based approach outperforms a RL algorithm on the state set. We compared our algorithm with a bandit approach and investigated the evaluation constant in the evaluation function and the reward prior distribution. We observe that the evaluation constant has no significant impact on the average performance of the algorithm, but it does influence the minimum and maximum obtained regret. Having an appropriate prior distribution ensures that the algorithm fits the proper reward distribution and this results in more consistent performance of the C-PSRL. However, the key to good performance of the C-PSRL is the cluster assignment. Therefore, the selection of the clustering length is crucial.

To conclude, reducing the state space of a MDP reduces the computational complexity of the problem significantly. If such a reduced latent space exists and we are able to learn this structure, this will be beneficial to the agent. If the underlying cluster structure can be described by a BMC combining the clustering algorithm of [40] with the reduced BMC–MDP model yields a significant increase in performance because we can aggregate data which results in a better estimate of the problem parameters. Summarizing, a combined approach of clustering and MDP solving on a reduced space is a viable approach. If we can relax the assumptions of this method to be applicable to general cluster based MDPs, this will result in a significant computational reduction. In the future this will make RL on high dimensional state spaces feasible.

6

Recommendations and future work

This thesis is a first study on the possibilities to apply a clustering algorithm for Block Markov Chains in the area of Reinforcement Learning. The basis of this study is built upon various assumptions to able to get a first look into the behaviour of a combined approach in which we leverage clustering. By nature of assuming, this also implies that there are various possibilities to expand upon. We now will briefly discuss these opportunities where we will not only state them but also provide some initial thoughts.

6.1 | Clustering Algorithm

The cluster assignment is of fundamental importance to the performance of any clustering based Reinforcement Learning approach in which the data samples for all states belonging to a cluster are aggregated. In this thesis we use the algorithm to cluster on a trajectory generated by a single BMC developed in [40]. We will now describe further research opportunities on the cluster algorithm.

In Section 4.1 we highlighted possibilities of transition dynamics that the BMC–MDP model can describe. However, if we want to use different transition dynamics for the total BMC–MDP problem it is necessary that the clustering methodology scope of applications is widened beyond strict BMCs. We propose to look at the possibility to cluster on a trajectory generated by a single transition matrix that is not exactly aBMC. So a possible research direction is analysing the performance or the extension of the clustering algorithm on transition matrix where there is some sort of block structure but this structure is not exactly the current definition of a BMC. For example: the transition matrix could have similar states that can be seen as blocks but are not identical to each other in terms of behaviour so there are small derivations in the transition probabilities, or we take a transition matrix that includes self jumps, or an irreducible transition matrix where multiple states are not reachable from a single state.

In Section 3.5 and Section 3.6 we have demonstrated that the cluster algorithm from [40] is also capable of clustering on a single trajectory generated by an alternating Block Markov Chain. However, in this first approach, additional data that is available in the MDP setting is not taken into account. It should be possible to change the clustering algorithm so that the information of multiple transition matrices for every action is taken into account. This potentially results in a different spectral clustering step on multiple matrices for every action; inspiration may be found in [16]. The novel research would be to maintain some form of the improvement step of [40]. The improvement step is likely vital to generate a superior cluster assignment compared to a clustering methodology just containing a spectral method. This new cluster algorithm may potentially results in a cluster assignment that can be proven to be accurate, or even exact, on a trajectory length where we utilise the dependencies of the state trajectory on the action trajectory in a MDP, using similar strategies to [40].

6.2 | Cluster based MDP

Besides the clustering algorithm there are also research possibilities in the combined approach of clustering and optimization of the BMC–MDP.

The main objective for this type of research is to derive a regret bound for this type of algorithm so that we can make quantitative statements on the performance, as well as compare this approach to

other approaches that tackle these types of optimization problems. There are opportunities to derive a regret bound for this algorithm structure because the regret of randomly picking actions in a MDP is linear and therefore regret bounds for the initial stage should be possible to derive. Because we know [40] the clustering algorithm's performance bounds. We then have PSRL with an initial estimate of the parameters as we already have an action, reward, and state trajectory from the initial stage. We can possibly use the current regret bound conditioned with this initialization of the estimates and conduct a regret bound for the total approach. We have to remark that we then still have to adjust for the correctness of the cluster assignment at the clustering length as this gives a bound on the theoretical number of correctly picked actions that is possible. In that sense a Probably Approximately Correct (PAC) bound may be researched for this type of algorithm as you can consider the convergence speed to the optimal action of the algorithm but with the condition that we have a cluster assignment, the accuracy of which will determines the speed. This is because misclassified states give an error to the estimated parameters and if the number of misclassified states goes to zero this error goes to zero. This could potentially also lead to a maximum bound on the number of misclassified states for which the algorithm can converge to an optimal solution at all.

In our algorithm structure, here we have made the explicit choice to cluster at the clustering length and then run a RL algorithm on the reduced underlying MDP with K states (clusters). A possibility is to investigate an algorithm with a different structure, namely an algorithm where we cluster multiple times. Here the cluster moments could be determined in a similar fashion as the re-evaluation of the policy. So we also use an increasing length between cluster moments and in this case similar argumentation holds for increasing the length between these intervals as to why we choose to take increasing intervals after which we re-evaluate the policy. However this kind of algorithm is probably more difficult to analyse if the objective is to give a regret analysis. The change in cluster assignment and correctness of this assignment is critical for the total regret bound that likely requires bounds on the number of correctly classified states at every cluster moment.

In the C-PSRL we pick the actions uniformly at random during the initial stage. If the clustering length is short compared to the number of cluster action pairs, for example when there are a large number of actions, we could potentially end up with an unbalanced number of samples for every action given the cluster. The algorithm ensures that afterwards we try these actions to explore them further in the third stage, but we perhaps could prevent this with a different action selection criteria in the initial stage to balance out the initial estimates.

Bibliography

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*, vol. 5. Athena Scientific Belmont, MA, 1996.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, draft, second ed., 2017.

[3] L. S. Shapley, "Stochastic games," *Proceedings of the National Academy of Sciences*, vol. 39, no. 10, pp. 1095–1100, 1953.

[4] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1 ed., 1957.

[5] P. Kormushev, S. Calinon, and D. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.

[6] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[7] I. Szita and A. Lörincz, "Learning Tetris Using the Noisy Cross-Entropy Method," *Neural Computation*, vol. 18, no. 12, pp. 2936–2941, 2006.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[9] V. Firoiu, W. F. Whitney, and J. B. Tenenbaum, "Beating the World's Best at Super Smash Bros. with Deep Reinforcement Learning," *arXiv preprint arXiv:1702.06230*, 2017.

[10] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[11] O.-A. Maillard and S. Mannor, "Latent Bandits.," in *International Conference on Machine Learning*, pp. 136–144, 2014.

[12] A. Lazaric, E. Brunskill, *et al.*, "Sequential transfer in multi-armed bandit with finite set of models," in *Advances in Neural Information Processing Systems*, pp. 2220–2228, 2013.

[13] L. Li, T. J. Walsh, and M. L. Littman, "Towards a Unified Theory of State Abstraction for MDPs.," in *ISAIM*, 2006.

[14] E. Even-Dar and Y. Mansour, "Approximate equivalence of Markov decision processes," in *Learning Theory and Kernel Machines*, pp. 581–594, Springer, 2003.

[15] R. Ortner, "Adaptive aggregation for reinforcement learning in average reward Markov decision processes," *Annals of Operations Research*, vol. 208, no. 1, pp. 321–336, 2013.

[16] K. Azizzadenesheli, A. Lazaric, and A. Anandkumar, "Reinforcement Learning in Rich-Observation MDPs using Spectral Methods," *CoRR*, vol. abs/1611.03907v4, 2018.

[17] J. Sanders, A. Proutière, and S.-Y. Yun, "Clustering in Block Markov Chains," *arXiv preprint arXiv:1712.09232v2*, 2018.

[18] O. Sigaud and O. Buffet, *Markov decision processes in artificial intelligence*. John Wiley & Sons, 2013.

[19] W. R. Thompson, "On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[20] H. Robbins, “Some aspects of the sequential design of experiments,” *Bulletin of the American Mathematical Society*, vol. 58, no. 5, pp. 527–535, 1952.

[21] T. Lai and H. Robbins, “Asymptotically efficient adaptive allocation rules,” *Advances in Applied Mathematics*, vol. 6, no. 1, pp. 4 – 22, 1985.

[22] D. A. Berry and B. Fristedt, *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, London, 1985.

[23] T. Lattimore and C. Szepesvári, “Bandit algorithms,” *preprint*, 2018.

[24] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” *Machine Learning*, vol. 47, pp. 235–256, May 2002.

[25] O.-C. Granmo, “Solving two-armed bernoulli bandit problems using a bayesian learning automaton,” *International Journal of Intelligent Computing and Cybernetics*, vol. 3, no. 2, pp. 207–234, 2010.

[26] P. A. Ortega and D. A. Braun, “A minimum relative entropy principle for learning and acting,” *Journal of Artificial Intelligence Research*, vol. 38, pp. 475–511, 2010.

[27] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich, “Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft’s bing search engine,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 13–20, 2010.

[28] D. Russo, B. V. Roy, A. Kazerouni, and I. Osband, “A Tutorial on Thompson Sampling,” *CoRR*, vol. abs/1707.02038, 2017.

[29] E. Kaufmann, N. Korda, and R. Munos, “Thompson Sampling: An Asymptotically Optimal Finite Time Analysis,” *ArXiv e-prints*, May 2012.

[30] M. Kearns and S. Singh, “Near-optimal reinforcement learning in polynomial time,” *Machine learning*, vol. 49, no. 2-3, pp. 209–232, 2002.

[31] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1st ed., 1994.

[32] L. Kallenberg, “Markov Decision Processes,” *URL https://www.math.leidenuniv.nl/ kallenberg/Lecture-notes-MDP.pdf*, 2017.

[33] D. A. Levin and Y. Peres, *Markov chains and mixing times*, vol. 107. American Mathematical Soc., 2017.

[34] T. Jaksch, R. Ortner, and P. Auer, “Near-optimal regret bounds for reinforcement learning,” *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1563–1600, 2010.

[35] P. L. Bartlett and A. Tewari, “REGAL: A regularization based algorithm for reinforcement learning in weakly communicating MDPs,” in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pp. 35–42, AUAI Press, 2009.

[36] I. Osband, *Deep exploration via randomized value functions*. PhD thesis, Stanford University, 2016.

[37] A. L. Strehl and M. L. Littman, “An analysis of model-based interval estimation for Markov decision processes,” *Journal of Computer and System Sciences*, vol. 74, no. 8, pp. 1309–1331, 2008.

[38] M. Strens, “A Bayesian framework for reinforcement learning,” in *ICML*, pp. 943–950, 2000.

[39] I. Osband, D. Russo, and B. Van Roy, “(More) efficient reinforcement learning via posterior sampling,” in *Advances in Neural Information Processing Systems*, pp. 3003–3011, 2013.

[40] J. Sanders and A. Proutière, “Optimal Clustering Algorithms in Block Markov Chains,” *ArXiv e-prints*, Dec. 2017.

[41] I. Szita and C. Szepesvári, “Model-based reinforcement learning with nearly tight exploration complexity bounds,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 1031–1038, 2010.

[42] I. Osband and B. Van Roy, “Why is posterior sampling better than optimism for reinforcement learning?,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2701–2710, JMLR. org, 2017.