Advanced Reinforcement Learning Algorithms for Multi-Armed Bandit Problems

Francisco Robledo Relaño

PhD Thesis

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

UNIVERSITÉ
DE PAU ET DES
PAYS DE L'ADOUR

PhD Thesis

Advanced Reinforcement Learning Algorithms for Multi-Armed Bandit Problems

Francisco Robledo Relaño

Supervisors
Urtzi Ayesta
Florin Avram

August 30, 2024

Acknowledgements

There are many people I would like to thank for their support and help during the development of my PhD.

First and foremost, I would like to express my deepest gratitude to my PhD advisor, Dr. Urtzi Ayesta, for his exceptional guidance, support, and advice throughout my research journey. His expertise in the field of Reinforcement Learning and Markov Decision Processes has been invaluable to me. His patience, encouragement, and constructive feedback have significantly contributed to my professional growth and the successful completion of this dissertation.

I would also like to extend my heartfelt thanks to Dr. Florin Avram for his support during my PhD.

I am also deeply grateful to Dr. Konstantin Avrachenkov and Vivek Borkar for their support and contributions to my research. Their expertise and advice have been crucial in overcoming many challenges during my PhD.

Last but certainly not least, I would like to express my deepest appreciation to my family for their unwavering support, love, and encouragement throughout my PhD. Their belief in me and their understanding have been my anchor during the most challenging times. To my parents, for their unconditional love and for instilling in me the value of education;

Without the collective support and encouragement of all these individuals, this thesis would not have been possible. Thank you all.

Abstract

This thesis presents advances in Reinforcement Learning (RL) algorithms for resource and policy management in Restless Multi-Armed Bandit (RMAB) problems. We develop algorithms through two approaches in this area. First, for problems with discrete and binary actions, which is the original case of RMAB, we have developed QWI and QWINN. These algorithms compute Whittle indices, a heuristic that decouples the different RMAB processes, thereby simplifying the policy determination. Second, for problems with continuous actions, which generalize to Weakly Coupled Markov Decision Processes (MDPs), we propose LPCA. This algorithm employs a Lagrangian relaxation to decouple the different MDPs.

The QWI and QWINN algorithms are introduced as two-timescale methods for computing Whittle indices for RMAB problems. In our results, we show mathematically that the estimates of Whittle indices of QWI converge to the theoretical values. QWINN, an extension of QWI, incorporates neural networks to compute the Q-values used to compute the Whittle indices. We establish mathematically the local convergence properties of the neural network used in QWINN. Our results show how QWINN outperforms QWI in terms of convergence rates and scalability.

In the continuous action case, the LPCA algorithm applies a Lagrangian relaxation to decouple the linked decision processes, allowing for efficient computation of optimal policies under resource constraints. We propose two different optimization methods, differential evolution and greedy optimization strategies, to efficiently handle resource allocation. In our results, LPCA shows superior performance over other contemporary RL approaches.

Empirical results from different simulated environments validate the effectiveness of the proposed algorithms. These algorithms represent a significant contribution to the field of resource allocation in RL and pave the way for future research into more generalized and scalable reinforcement learning frameworks.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Overview of Reinforcement Learning and MDPs

1.1.1 Reinforcement Learning: An Introduction

The concept of learning through interaction is fundamental to our understanding of both natural and artificial intelligence. From the earliest moments of life, animals, including humans, engage with their environment in a continuous process of exploration and adaptation. This interaction-driven learning plays a key role in shaping behaviors and capabilities.

Consider a child learning to navigate the world: through trial and error, they discover the consequences of their actions. They learn that touching a hot surface causes pain, and that listening to a lullaby often precedes sleep. This process of acquiring knowledge through active engagement, where outcomes influence subsequent actions, reflects the basic principle of learning theories. It highlights a critical aspect of cognitive development-the ability to learn from the outcomes of interactions, rather than from passive observation or instruction alone.

This learning paradigm, deeply rooted in the biological processes of natural organisms, has inspired the development of artificial learning systems and led us to the diverse field of machine learning.

In the field of machine learning, this natural learning process has been abstracted and translated into computational models. Machine learning includes several paradigms, each of which reflects a different aspect of learning in the natural world: Supervised Learning, Unsupervised Learning, and Reinforcement Learning.

Supervised Learning is similar to the way humans learn from instruction. In this case, the learning algorithm is presented with labeled examples, which it uses to learn a mapping from inputs to outputs. This is similar to a child learning to recognize animals from a book where each picture is labeled. Supervised Learning excels at tasks where the relationship between input data and output labels is clear, such as image classification and regression problems. Unsupervised Learning, on the other hand, involves finding patterns or structures in unlabeled data. It's like a child observing the world and identifying patterns without being explicitly told what to look for. This paradigm is essential in scenarios where the underlying structure of the data is unknown or where explicit labels are not available. It's used in clustering, anomaly detection, and dimensionality reduction.

Reinforcement Learning (RL), the focus of this thesis, is inspired by the way organisms learn from the consequences of their actions. In RL, an agent learns to make decisions by interacting with its environment. The agent receives rewards or penalties based on its actions, which induce it to adjust its behavior to achieve better outcomes over time. Going back to the child example, this would be equivalent to receiving rewards such as toys or treats for good behavior and punishments for bad behavior. This learning process is not about finding patterns in existing data, but about discovering a strategy or policy through the consequences of actions taken in a dynamic environment.

Reinforcement Learning is characterized by its similarity to the natural learning processes observed in animals and humans. Reinforcement Learning embodies the essence of learning by doing, a principle central to cognitive development and behavioral adaptation. In RL, the agent's ability to experiment, receive feedback, and adjust its actions accordingly is critical. This mirrors the way a child learns to walk or an animal learns to survive in its environment. It also differs from other types of machine learning in the need to balance exploration and exploitation of information. Our agents need to make the decisions that they know will maximize their reward, but to do so, they must first explore the different types of actions available to learn which ones are best. In other words, they must balance the "exploitation" of information they already know with the "exploration" of new information. Moreover, these agents cannot focus on either of these processes without ultimately being detrimental to the task at hand: they must perform a variety of actions while progressively focusing on those that are most beneficial.

1.1.2 Historical introduction to Reinforcement Learning

The history of reinforcement learning (RL) is a tapestry woven from two distinct threads: optimal control and behavioral (trial-and-error) learning. Understanding this evolution provides insight into RL's unique position in the Machine Learning landscape.

The concept of optimal control emerged in the 1950s, focusing on the design of controllers to optimize the behavior of dynamical systems. A key figure in this field was Richard Bellman, who in 1957 introduced the concept of a value function or "optimal return function", later known as the Bellman equation, to represent the state of a dynamical system. This equation laid the foundation for dynamic programming [1] and Markov Decision Processes (MDPs) [2], foundational elements in RL. The latter will be further developed in the following subsection.

Despite the challenge of the "curse of dimensionality", where the problem complexity grows exponentially with the number of states, dynamic programming remains an important approach for solving general stochastic optimal control problems. The integration of dynamic programming with learning paradigms became well-defined in the late 1980s, highlighted by Watkins' pioneering work on reinforcement learning using the MDP formalism [3].

Simultaneously, the field of behavioral learning, based in trial-and-error methods, was developing. Edward Thorndike's "Law of Effect" [4], a principle that states that actions leading to satisfying states are likely to be repeated, was an early cornerstone. Influencing various learning theories and experimental methods, such as Skinner's [5], Thorndike's work laid the groundwork for future research in behavioral learning.

In the 1940s, Alan Turing's notion of a "pleasure-pain system" [6] paralleled Thorndike's ideas and foreshadowed modern RL concepts. In the 1960s, however, the focus in AI shifted to supervised learning, causing a temporary decline in trial-and-error research. Notable exceptions included Marvin Minsky's [7] discussions of issues relevant to trial-and-error learning, and the development of "learning automata" approaches [8], analogous to the k-armed bandit problem, by analogy to a slot machine, except with k levers instead of one.

Harry Klopf's works [9, 10, 11] in the 1970s reinvigorated interest in trial-and-error learning by emphasizing the need for algorithms to guide their environment toward desired outcomes. This perspective helped to delineate the differences between supervised and reinforcement learning.

The concept of temporal difference (TD) learning emerged as a unifying force between the two strands of optimal control and behavioral learning. Its origins can be traced back to animal learning psychology, specifically the idea of secondary reinforcers. Secondary reinforcers are stimuli that, when associated with primary reinforcers, acquire the ability to produce similar effects.

Temporal difference learning was first proposed by Arthur Samuel [12] in 1959 as part of his checker playing program. Following Samuel's initial contributions, research in TD learning experienced a period of reduced activity until the 1980s, when Harry Klopf's work [13] brought renewed attention to TD methods and set the stage for significant developments.

In the late 1970s and 1980s, Richard S. Sutton and Andrew G. Barto [14] extended Klopf's theories into a psychological model of classical conditioning based on TD learning. Their work led to the development of the actor-critic architecture, which combines TD and trial-and-error learning, as seen in its application to the pole-balancing problem [15]. TD was finally formalized in Sutton's 1988 paper [16], which introduced the TD(λ) algorithm and some of its convergence properties.

The convergence of temporal difference learning and optimal control was fully realized in 1992 with the development of Q-learning by Christopher Watkins [17]. This represented a major milestone in the history of RL, combining the strengths of dynamic programming with the principles of trial-and-error learning. Paul Werbos [18] also contributed significantly to this integration, arguing for the convergence of trial-and-error learning and dynamic programming. His arguments further solidified the foundation of RL as an interdisciplinary field, synthesizing concepts from control theory, psychology, and computer science.

The integration of function approximation methods, especially neural networks [19], in the 1990s and 2000s marked another leap forward. This integration addressed the challenge of scalability in RL, allowing the application of RL algorithms to problems with large or continuous state spaces. The seminal paper "Playing Atari with Deep Reinforcement Learning" by Mnih et al. in 2013 [20] exemplified this advancement, demonstrating the potential of combining deep learning with RL.

In recent years, RL has made remarkable progress, driven by advances in computational power, algorithmic efficiency, and cross-disciplinary integration. The field has expanded beyond its traditional domains, finding applications in areas such as robotics, healthcare, finance, and autonomous systems [21, 22, 23]. The development of algorithms capable of handling complex, real-world environments with high-dimensional data is a major focus of contemporary research.

Current trends also include the exploration of multi-agent RL [24, 25, 26], where the presence of multiple learners introduces new challenges and dynamics. The interplay between RL and other areas of machine learning, such as unsupervised and supervised learning, is another growing area of interest, leading to more robust and versatile learning models.

1.1.3 An introduction to Markov Decision Processes

A Markov Decision Process (MDP) is a fundamental concept in reinforcement learning that provides a formal mathematical framework for modeling decision making in environments with stochastic outcomes that are influenced by the actions of a decision maker. The MDP concept, introduced by Richard Bellman [2], has been essential in the study and development of RL algorithms.

At its core, an MDP is a stochastic control process, which means that it models scenarios where events unfold overtime, with randomness playing a significant role in the progression of these events. This framework is particularly well-suited for decision-making situations where outcomes are partly under the control of a decision maker (such as an RL agent) and partly random.

The theoretical foundation of MDPs is rooted in the concept of Markov chains, named after the Russian mathematician Andrey Markov [27]. A Markov chain is a stochastic model that describes a sequence of possible events, where the probability of each event depends only on the state reached in the previous event.

Two types of Markov chains can be distinguished: discrete-time Markov chains (DTMC) and continuous-time Markov chains (CTMC). Discrete-time chains perform discrete transitions at regular time intervals, while continuous-time chains can perform transitions at any continuous time. The key property of Markov chains, and by extension MDPs, is the Markov property or "memorylessness". This property implies that future states of the process depend only on the current state, not on the sequence of events that preceded it, and therefore predictions made with knowledge of the previous state are as good as those made with knowledge of all previous states. This can be thought of as a system where "what happens next depends only on the current state of affairs".

In an MDP, at each time step the system is in a particular state s, and the decision maker has the option of choosing an action a from those available in that state. The system responds to this action by transitioning to a new state s' at the next time step. This transition is not necessarily deterministic, but probabilistic with a transition probability $P(s'|s,a)$, which is influenced by the chosen action. Along with this transition, the decision maker receives a reward $R(s,a)$, which represents the immediate payoff of performing the action in this state.

In the context of RL, MDPs provide a structured way to formalize the interaction between an agent (decision maker) and its environment. They provide a clear framework for defining the agent's goals (maximizing cumulative rewards), the challenges it faces (dealing with stochastic outcomes), and the strategies it must develop (learning optimal actions in given states).

MDPs are central to many RL algorithms, providing the mathematical foundation for the learning process. Whether an agent is navigating a maze, playing a game, or managing resources, the MDP framework helps formalize the problem in a way that facilitates the application of RL techniques.

1.2 Significance and Applications of RL in MDPs

Markov Decision Processes (MDPs) provide a powerful framework for modeling decision-making in environments where outcomes are influenced by both randomness and the actions of a decision-maker. The significance of Reinforcement Learning (RL) in solving MDPs, especially when the parameters of these processes—such as transition probabilities and reward functions—are unknown, cannot be overstated. RL's ability to learn and optimize in uncertain and dynamic environments makes it a critical tool for navigating the complexities inherent in MDPs.

One of the fundamental challenges in applying MDPs to real-world problems is the lack of complete knowledge about the environment. In many scenarios, the decision-maker does not have access to the transition probabilities that dictate how the environment will respond to its actions, nor does it know the reward function that evaluates the outcomes of its decisions. This is where RL shines, offering methodologies for the agent to learn these unknown parameters through iterative interaction with the environment: the agent performs an action, observes the resulting state and the reward obtained, and updates its knowledge of the MDP based on this experience, learning to predict the consequences of its actions. This allows the RL agent to learn an optimal policy: a strategy that dictates the best action to perform in each state to maximize such rewards in the long run.

Consider, for example, a navigation problem where a robot is tasked with finding the shortest path to a goal in a maze. The maze can be modeled as an MDP, where each location is a state, and moving from one location to another is an action. The transition probabilities (the likelihood of moving successfully from one state to another) and the reward function (which can be modeled as the immediate cost or benefit of moving to a new state) are initially unknown to the robot. As the robot explores the maze, it learns from each movement's success or failure, gradually building an understanding of the maze's layout (system's dynamics such as the transition probabilities and the state whole state space) and the consequences of its actions (the rewards). Through reinforcement learning, the robot develops a policy that efficiently guides it to the goal, minimizing the total travel time or distance.

Another illustrative example is in the domain of healthcare, where treatment strategies can be modeled as MDPs. Here, states could represent different health conditions of a patient, actions could correspond to various treatment options, and rewards could be based on treatment outcomes. Given the complexity and uncertainty of human health, transition probabilities (how a patient's condition will change following a treatment) and reward functions (the effectiveness of treatments) are often unknown or highly uncertain. RL can be employed to learn optimal treatment policies by iteratively experimenting with different treatment strategies and observing patient outcomes, thereby learning to maximize patient health over time.

The management of renewable resources, such as a fishery, is a further useful example. Here, the states could represent the size of the fish population, the actions could be different levels of fishing effort, and the rewards could reflect the balance between immediate profit from fishing and the long-term sustainability of the fishery. The transition probabilities, which indicate how the fish population evolves in response to fishing efforts, are initially unknown and potentially complex, influenced by numerous environmental factors. By applying RL, a policy can be developed that optimizes the long-term yield of the fishery, ensuring its sustainability while maximizing profits. Through trial and error, the RL agent learns which fishing efforts lead to sustainable practices, effectively managing the resource without explicit knowledge of the biological dynamics at play.

In the financial sector, portfolio management can also be framed as an MDP problem, where the states represent different market conditions, actions are investment choices, and rewards are financial returns. The stochastic nature of financial markets means that both the transition probabilities (how market conditions evolve) and reward functions (returns on investments) are uncertain and dynamic. RL techniques can learn from historical data to predict market trends and develop investment strategies that maximize long-term returns, adapting to new information as it becomes available.

Through its capacity to learn from interaction and adapt to changing environments, RL empowers decision-makers to tackle a wide array of complex, dynamic problems.

1.3 Thesis Objective and Scope

The previous examples illustrate the broad applicability of Reinforcement Learning (RL) for solving Markov Decision Processes (MDPs) in various domains. However, although RL has shown remarkable success in many applications, in most cases this has been achieved by using huge amounts of data and computational resources. This thesis aims to explore the complex realm of Markov Decision Processes (MDPs), with a special focus on multi-armed bandit processes, restless multi-armed bandit problems, and weakly coupled MDPs with continuous actions. These specialized MDPs represent critical areas of research within the broader field of reinforcement learning, and offer unique opportunities to obtain more efficient RL algorithms by exploiting the specific structure of these problems.

Chapter 2 will provide a detailed introduction to MDPs and multi-armed bandit processes. Among these, restless multi-armed bandit problems and Weakly Coupled MDPs stand out for their practical importance. These problems exemplify situations where there are multiple MDPs that, while not necessarily interacting with each other, share resources from which they can perform actions, leading to situations where an action in a given MDP affect the available actions in the other. These environments not only evolve influenced by the agent actions but also independently, increasing the complexity of devising optimal strategies.

Chapter 3 focuses on the application of classical Reinforcement Learning algorithms such as Q-learning and SARSA to MDPs, as well as Neural Networks and Deep Reinforcement Learning. These algorithms provide a solid foundation for understanding the challenges presented by these problems, and the basis for the algorithms proposed in this thesis.

The contributions of this thesis are divided into two major sections: In Chapter 4, we will explore the models for Restless Multi-Armed Bandit Problems introduced in detail in Section 2.2.3, starting with the QWI (Section 4.1), first introduced in [28] and later expanded in [29], and QWINN [29] (Section 4.2) algorithms, along with an analysis of their numerical results (Section 4.3), including the description of the environments employed, metrics and comparison.

In Chapter 5 we will focus on models for Weakly Coupled MDPs with continuous actions, introduced in 2.2.4. In this section we will introduce LPCA [30], a learning algorithm to solve these Weakly coupled MDPs with continuous actions (section 5.2). We will compare the approach of both types of heuristics to these types of problems and show experimental results along with descriptions of the environments used and evaluation metrics (section 5.3).

Finally, we will give a summary of the contributions and results of this thesis (section 6), along with challenges and areas for further research.

Chapter 2

Background and Literature Review

2.1 Fundamentals of Markov Decision Processes

2.1.1 Markov Processes

As we have seen in section 1.1.3, Markov Decision Processes (MDPs) serve as a foundational framework in reinforcement learning, offering a formal description of environments where decision-making under uncertainty is central. At the heart of this framework lies the concept of Markov processes, also known as Markov chains. Through the following sections, we will explore the fundamental principles of Markov processes and Markov Reward Processes, and Markov Decision Processes. A more detailed explanation of these concepts can be found in [31].

A Markov process characterizes an environment that is fully observable, where the current state encapsulates all the necessary information to describe the process at any given time. This feature simplifies the decision-making landscape, as there is no need to consider the previous history of states to predict the future. Instead, the present state offers all the information from the environment, making past states irrelevant for future decision-making. This principle is present in most reinforcement learning problems, allowing them to be formalized as MDPs.

This is the defining characteristic of a Markov process, the Markov property, which can be succinctly captured by the statement, “The future is independent of the past given the present”. Mathematically, this property is expressed as

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, \dots, s_t),$$

indicating that the probability of transitioning to the next state s_{t+1} depends solely on the current state s_t and not on the sequence of states that preceded it. This property implies that the state is *Markovian*, embodying all relevant information from the history of the process. Once the state is known, the history leading up to that state can effectively be discarded, as it offers no additional predictive power regarding future states.

The transition between states in a Markov process is quantified by transition probabilities. For a transition from state s to state s', the probability is denoted as $P_{ss'} = P(S_{t+1} = s'|S_t = s)$. These transition probabilities are organized into a transition matrix P, with each element P_{ij} representing the probability of moving from state i to state j. This matrix is structured as follows:

$$P = \begin{pmatrix} P_{11} & \dots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \dots & P_{nn} \end{pmatrix}$$

One of the key properties of this matrix is that each row sums to 1, reflecting the certainty that the process will transition from the current state to some state in the next time step.

A Markov process, in essence, is a memoryless random process, a sequence of random states $s_1, s_2, \dots$ that adhere to the Markov Property. Formally it is defined by the tuple $\langle \mathcal{S}, P \rangle$, where:

- **State space** $\mathcal{S}$: The set of all possible states or the state space. The state of the system at time t is denoted as $s(t)$.
- **Transition probability matrix** P: The transition probability matrix that governs the dynamics of the process. Each element $P_{ss'} = \mathbb{E}P(S_{t+1} = s'|S_t = s)$ represents the probability of transitioning from state s to state s'.

2.1.2 Markov Reward Processes

Expanding on the foundational concept of Markov chains, the introduction of Markov Reward Processes (MRPs) incorporates the crucial element of value into the equation, transforming the basic structure of Markov chains by adding rewards to the transitions between states. An MRP is defined by the tuple $\langle \mathcal{S}, P, R \rangle$, where:

- **State space** $\mathcal{S}$: The set of all possible states, representing the environment's observable states. The state of the system at time t is denoted as $s(t)$.
- **Transition probability matrix** P: The transition probability matrix, with $P_{ss'} = \mathbb{E}P(S_{t+1} = s'|S_t = s)$ indicating the probability of transitioning from state s to state s'.
- **Reward function** R: The reward function, $R_i = \mathbb{E}[R_{t+1}|S_t = s]$, defining the expected reward received when transitioning from state s.

Definition 2.1. *The return G_t is the total accumulated reward from time step t onwards.*

This concept introduces different optimality criteria for evaluating the performance of policies within this framework:

- **Expected total reward**: Defined as the sum of the expected rewards from time step t onwards:

  $$G = \lim_{t_{\max} \to \infty} \mathbb{E}\left[\sum_{t=0}^{t_{\max}-1} R_t\right].$$

  This criterion, however, can lead to infinite returns, as the sum of rewards can diverge [32].

- **Expected average total reward**: Defined as the averaged sum of the expected rewards from time step t onwards:

  $$G = \lim_{t_{\max} \to \infty} \mathbb{E}\left[\frac{1}{t_{\max}} \sum_{t=0}^{t_{\max}-1} R_t\right].$$

- **Expected total discounted reward**: Defined as the sum of the expected discounted rewards from time step t onwards:

  $$G = \lim_{t_{\max} \to \infty} \mathbb{E}\left[\sum_{t=0}^{t_{\max}-1} \gamma^t R_t\right],$$

  where $\gamma \in [0, 1)$ [33].

We will focus our discussion on the expected total discounted reward. This criterion prioritizes immediate rewards over delayed rewards, introducing a factor γ, known as the discount factor, which modulates the importance of future rewards. A γ close to 0 leads to a myopic evaluation, emphasizing immediate rewards, while a γ close to 1 enables a far-sighted evaluation, valuing future rewards more significantly.

The use of discounted rewards is justified by several reasons. First, it avoids the problem of potentially infinite rewards. Secondly, it can model the uncertainty regarding the continuation of the environment at the next decision instant, making it particularly relevant in scenarios where the decision horizon is uncertain. Lastly, in fields like finance, immediate rewards can be more valuable as they can earn interest over time. This approach mirrors the observed behavior in animals and humans, who often show a preference for immediate rewards over delayed ones.

Definition 2.2. *The state value function $V(s)$ of an MRP is defined as the expected return starting from state s:*

$$V(s) = \mathbb{E}[G_t | S_t = s].$$

This value function represents the long-term value of being in state s. The value function can be decomposed into two parts: the immediate reward R_{t+1} and the discounted value of the successor state $\gamma V(S_{t+1})$.

$$\begin{aligned}
V(s) =&\mathbb{E}[G_t|S_t = s] \\
=&\mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s] \\
=&\mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots)|S_t = s] \\
=&\mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
=&\mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s]
\end{aligned}$$

This decomposition leads to the Bellman Equation [34], which facilitates an intuitive understanding of the value function as a balance between immediate reward and the anticipated future benefits, encapsulated in the equation $V(s) = R(s) + \gamma \sum_{s' \in S} P_{ss'} V(s')$.

Given full knowledge of the reward function and the transition probability matrix, the Bellman equation can be expressed in matrix form and solved directly:

$$\begin{aligned}
V =&R + \gamma PV \\
(I - \gamma P)V =&R \\
V =&(I - \gamma P)^{-1} R
\end{aligned} \tag{2.1}$$

However, this computational approach has a complexity of $O(n^3)$ for n states, showing the challenge in solving these problems for large state spaces.

2.1.3 Markov Decision Processes

Markov Decision Processes (MDPs) extend the Markov Reward Processes (MRPs) framework by incorporating decisions into the model, effectively turning it into a comprehensive tool for decision-making under uncertainty. An MDP is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R \rangle$, where:

- **State space** $\mathcal{S}$: The set of all possible states, representing the environment's observable states. The state of the system at time t is denoted as $s(t)$.
- **Action space** $\mathcal{A}$: The set of all possible actions, representing the decisions that can be made by the agent. The action taken at time t is denoted as $a(t)$.
- **Transition probability matrix** P: The transition probability matrix, with $P_{ss'}^a = \mathbb{E}P(S_{t+1} = s'|S_t = s, A_t = a)$ indicating the probability of transitioning from state s to state s' under action a.
- **Reward function** R: The reward function, $R_s^a = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$, defining the expected reward received when taking action a in state s.

Central to an MDP is the concept of a policy π, a distribution over actions given states, defined as $\pi(a|s) = P(A_t = a|S_t = s)$. The policy dictates the behavior of the reinforcement learning (RL) agent, prescribing a specific action or set of actions to be taken in each state. Importantly, MDP policies are dependent solely on the current state, reflecting the Markov property's emphasis on the sufficiency of present information for decision-making.

Definition 2.3. *The state-value function $V_\pi(s)$ of an MDP is defined as the expected return starting from state s and following policy π:*

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]. \tag{2.2}$$

Definition 2.4. *The action-value function $Q_\pi(s, a)$ of an MDP is defined as the expected return starting from state s, taking action a, and thereafter following policy π:*

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]. \tag{2.3}$$

Both $V_\pi(s)$ and $Q_\pi(s, a)$ can be decomposed to reflect the immediate plus future rewards, represented by the discounted value of the successor state:

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s] \tag{2.4}$$

$$Q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{2.5}$$

These equations can be rewritten in terms of the transition probability matrix P and policy function π:

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s') \right)$$

$$Q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') Q_\pi(s', a')$$

Lastly, they can be expressed as functions of each other:

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) Q_\pi(s, a)$$

$$Q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s')$$

Definition 2.5. *The optimal state-value function $V^*(s)$ is defined as the maximum value function over all policies:*

$$V^*(s) = \max_\pi V_\pi(s).$$

Definition 2.6. *The optimal action-value function $Q^*(s, a)$ is defined as the maximum action-value function over all policies:*

$$Q^*(s, a) = \max_\pi Q_\pi(s, a).$$

These functions identify the best possible performance attainable from any state or state-action pair under the best policy. Given a partial ordering over the policies $\pi \geq \pi'$ if $V_\pi(s) \geq V_{\pi'}(s), \forall s \in S$, these value functions can be learned through iterative methods, such as value iteration or policy iteration, which iteratively update the value functions until convergence to the optimal $V^*(s)$ and $Q^*(s,a)$ is reached, or through learning methods such as Q-learning or SARSA, further introduced in Section 3.1.

Theorem 2.1. *For any MDP, there exists an optimal policy π^* that is better or equal to all other policies, $\pi^* \geq \pi, \forall \pi$.*

All optimal policies achieve an optimal value function $V_{\pi^*}(s) = V^*(s)$ and an optimal action-value function $Q_{\pi^*}(s,a) = Q^*(s,a)$.

The optimal policy can be found by maximizing over $Q^*(s,a)$:

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in A} Q^*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

However, due to the Bellman optimality equation's non-linearity, this process lacks a closed-form solution. This challenge has motivated the development of iterative solution methods, such as value iteration, policy iteration, Q-learning [17], and SARSA [35], each offering a pathway through the intricate landscape of MDPs towards optimal decision-making.

2.2 Multi-Armed Bandit Problems

2.2.1 Introduction To Multi-Armed Bandits

The multi-armed bandit (MAB) problem is a classic paradigm in the field of reinforcement learning and decision theory, originally described by Thompson [36] and further developed by Herbert Robbins [37] in 1952. This statistical decision model represents the dilemma faced by an agent attempting to make a series of decisions in an environment where it must simultaneously optimize its choices and gather information to inform future decisions. The metaphor of a gambler choosing which arm of a set of slot machines (or "bandits") to play captures the essence of the problem: each arm represents a different competing project or option, with its own distinct distribution of returns.

A core characteristic of the MAB problem is that the distribution of rewards from any given arm is influenced only when that arm is chosen. This means the outcomes from playing one arm are independent of the rewards obtained from other arms, isolating the impact of each decision to the specific context in which it is made. This property simplifies the decision-making process to a series of isolated choices, each with potentially different risks and rewards.

The relevance of MAB models extends far beyond theoretical exploration, finding practical application in scenarios such as clinical trials. In this example, the goal is to experiment with different treatments while minimizing patient risk and maximizing therapeutic outcomes. Each "arm" in this context represents a different treatment path, and the challenge lies in navigating the trade-off between exploring new treatments and exploiting known effective ones.

MAB models are fundamentally concerned with the optimal allocation of effort over time across a collection of projects or choices that change state randomly, depending on engagement. The activation rule is that at each discrete decision epoch, exactly one project must be selected, where each project is capable of being in one of a finite number of states. Projects not engaged maintain their states unchanged or *frozen*, obtaining no rewards under passivity: the challenge lies in activating projects in a manner that maximizes the total expected discounted reward over an infinite horizon.

After Thompson's [36] introduction of the problem, early academic exploration of MAB problems by Robbins [37], Bellman [38], and others laid the groundwork for a rich body of research. A significant milestone in the MAB problem was achieved by John C. Gittins, who introduced the concept of the priority-index rule [39], whereby an index is computed for each arm and each state, making the policy to activate the arm with the highest current index at each decision epoch. This approach was groundbreaking, offering a tractable solution to what was once considered an intractable challenge.

However, for more complex MAB extensions, the Gittins heuristic is not optimal. An example of this is when incorporating costs or delays when switching between projects, as is the case studied in [40].

2.2.2 MAB Model Formulation

In this section we present the MDP formulation and the concepts of the Multi-Armed Bandit problem.

In this model, an agent is tasked with allocating effort among a collection of N projects or arms, each characterized by its unique state space $\mathcal{S}_i$ and modeled as a Markov decision process. At each decision epoch t, for project i, a decision $a_i(t)$ is made at state $s_i(t)$ to either engage the project ($a_i(t) = 1$) or remain passive ($a_i(t) = 0$). Engagement results in an immediate reward $R_i(s_i)$ and a potential state transition to s_i', whereas passivity yields no reward and no state change.

The objective is formalized as maximizing the expected sum of discounted rewards over an infinite horizon:

$$\max \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(\mathbf{s}(t), \mathbf{a}(t))\right] \tag{2.6}$$

The standard MDP formulation for the MAB problem is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathbf{P}, \mathbf{R}, N \rangle$, where:

- **State space** $\mathcal{S}$: The combined state space of all projects, defined as the Cartesian product $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_N$, where $\mathcal{S}_i, 1 \leq i \leq N$ is the state space of project i. The state of the system at time t is denoted as $\mathrm{s}(t) = (s_1(t), s_2(t), \ldots, s_N(t))$, with $s_i(t)$ the state of project i at time t.

- **Action space** $\mathcal{A}$: The set of available actions at each decision epoch, defined as
$$\mathcal{A} = \left\{ \mathbf{a} = (a_1, \ldots, a_N) : \sum_{i=1}^{N} a_i = 1, a_i \in \{0, 1\} \right\},$$
where $\mathbf{a}(t) = (a_1(t), a_2(t), \ldots, a_N(t))$ is the action vector at time t, with $a_i(t)$ the action for project i at time t. Under action $a_i = 1$, bandit i is active, while under $a_i = 0$, it remains passive.

- **Transition probabilities** $\mathbf{P}$: Whenever a bandit i is active with action $a_i = 1$, it evolves according to its transition probability matrix:
$$P(s_i(t+1) = s' | s_i(t) = s, a_i(t) = 1) = P_i^1(s, s'), s, s' \in S_i$$
whereas for the passive bandits with $a_i = 0$, it remains in state s:
$$P(s_i(t+1) = s | s_i(t) = s, a_i(t) = 0) = P_i^0(s, s'), s, s' \in S_i$$
with
$$P_i^0(s, s') = \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{otherwise} \end{cases}$$
Under the whole system representation with $\mathrm{s} \in \mathcal{S}$ and action $\mathbf{a} \in \mathcal{A}$, the transition probability matrix is given by:
$$\mathcal{P}^{\mathbf{a}}_{\mathbf{s},\mathbf{s}'} = \prod_{i=1}^{N} P_i^{a_i}(s_i, s_i'), \forall \mathrm{s}, \mathrm{s}' \in \mathcal{S}, \mathbf{a} \in \mathcal{A}$$
We define $\mathbf{P}$ as the collection of transition matrices $\mathcal{P}^{\mathrm{a}}$, $\mathbf{P} = \{\mathcal{P}^{\mathbf{a}}_{\mathbf{s},\mathbf{s}'} : \mathrm{s}, \mathrm{s}' \in \mathcal{S}, \mathbf{a} \in \mathcal{A}\}$, where $\mathcal{P}^{\mathbf{a}}_{\mathbf{s},\mathbf{s}'}$ represents the probability of transitioning from state s to state s' under action $\mathbf{a}$.

- **Reward function** $\mathbf{R}$: The reward function is defined as the expected reward received when transitioning from state s to state s' under action $\mathbf{a}$. It is bounded and stationary and defined as $R_i : S_i \times A_i \to \mathbb{R}$ for all i. Given an active transition from state $s_i(t) = s$ to $s_i(t+1) = s'$, occurs on arm i at time t, a discounted reward $\gamma^t R_i(s, a)$ is obtained. Passive arms yield no reward.

- **Number of projects** N: The number of MDPs or projects, each with its unique state space S_i.

The optimal value function $V(\mathbf{s})$ represents the maximum expected return achievable from any initial state s, governed by the optimality equations:

$$V(\mathbf{s}) = \max_{\mathbf{a}\in\mathbf{A}} \left\{ R(\mathbf{s},\mathbf{a}) + \gamma \sum_{\mathbf{s}'\in\mathbf{S}} P^{\mathbf{a}}_{\mathbf{s},\mathbf{s}'} V(\mathbf{s}') | \sum_{i=1}^{N} a_i = 1 \right\}, \mathbf{s} \in \mathbf{S} \tag{2.7}$$

However, this formulation faces the *curse of dimensionality*, as the problem size grows exponentially with the number of projects, making, for a long time, this problem intractable.

A pivotal solution to this challenge is characterized by a dynamic allocation index, or Gittins index, defined for each state of each project. For any state $s_i \in S_i$, the index $\lambda_i(s_i)$ is defined as:

$$\lambda_i(s_i) = \sup_{\tau>0} \left\{ \frac{\mathbb{E}[\sum_{t=0}^{\tau-1} \gamma^t R(s_i(t))]}{\mathbb{E}\sum_{t=0}^{\tau-1} \gamma^t} \right\} \tag{2.8}$$

where τ is a stopping time that satisfies:

$$\tau(s_i) = \min\{t : \lambda_i(s_i(t)) < \lambda(s_i)\} \tag{2.9}$$

An important property of the Gittins index is that the supremum in (2.8) is achieved by (2.9) [39]:

Theory 2.1. *The supremum of* (2.8) *is achieved by* (2.9). *It is also achieved by any stopping time σ that satisfies:*

$$\sigma \leq \tau(s) \textit{ and } \lambda(s(\sigma)) \leq \lambda(s) \tag{2.10}$$

Thus, the optimal index policy is to select the project with the highest index at each decision step.

Theory 2.2. *There exists functions $\lambda_i(s_i(t)), 1 \ \leq i \leq N$ such that for any state $\mathrm{s}(t)$ the policy π^* which will activate the bandit process $m(t) = i$ which satisfies $\lambda(s_i(t)) = \max_{1\leq j\leq N} \lambda_j(s_j(t))$ is optimal. The function $\lambda_i(\cdot), 1 \leq i \leq N$ is calculated from the dynamics of process i alone.*

This approach simplifies the decision-making process by reducing the complex MAB problem to the computation of individual project indices $\lambda(s)$, which reflect the optimal reward rate attainable from each project when starting from its initial state $s_i(0) = s$.

Whittle's dynamic programming analysis [41] on the Gittins indices further expanded on Gittins' analysis by introducing an alternative arm, or "standard arm", that never changes state and provides a fixed reward. Whittle considered a modification of the multi-armed bandit project i, in which passivity is subsidized with a constant amount λ, so that $R_i(s, 0) = \lambda$. This leads to the optimal value function for a single-project sub-problem:

$$V_i(s, \lambda) = \max \left\{ R_i(s) + \gamma \sum_{s' \in S_i} P_i(s, s') V_i(s', \lambda); \lambda + \gamma V_i(s, \lambda) \right\}, s \in S_i \quad (2.11)$$

As λ varies from $-\infty$ to ∞, the set of states where the optimal action is the passive one increases monotonically from the empty set to the full set of the state space, with the Gittins Index emerging as the critical breakpoint: the unique value of λ that makes optimal both, the active and passive actions in state s.

2.2.3 Restless Multi-Armed Bandit Problems

The restless multi-armed bandit (RMAB) problem introduces a significant departure from the classical multi-armed bandit (MAB) framework studied in sections 2.2.1 and 2.2.2 by changing one of its core assumptions: that projects or arms remain in a static, *frozen* state when not engaged. This limitation restricts the applicability of the traditional MAB model to a broad array of real-world problems, where passive projects may evolve independently of active engagement, thus necessitating a more dynamic approach to decision-making.

Whittle [42] showcased this complexity with examples where projects continue to change states even in passivity. One classical example given by Whittle [42]:

> ... suppose m aircraft are trying to track the positions of n enemy submarines, where $m < n$, so that aircraft must change task from time to time if all submarines are to be monitored... The problem is to allocate this surveillance... While a submarine is under observation, information on its position... is being gained. While it is not, information is usually being lost, because the submarine will certainly be taking unpredictable evasive action.

Another example can be seen in deadline scheduling problems. In these scenarios, tasks or jobs must be completed by specific deadlines to avoid penalties or maximize rewards. The challenge lies in dynamically allocating limited resources to various tasks whose states can evolve over time, even when not actively worked on.

To address these challenges, in this subsection we will consider the case where M out of N projects must be selected for operation at each decision epoch. Unlike the traditional MAB model, both active and passive projects in RMAB evolve according to their respective transition rules with their corresponding transition probability matrices. This model is formalized as an infinite-horizon discounted criterion problem with a Markov Decision Process (MDP) represented by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathbf{P}, \mathbf{R}, N, M \rangle$.

- **State space $\mathcal{S}$**: The combined state space of all projects, defined as the Cartesian product $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_N$, where $\mathcal{S}_i, 1 \leq i \leq N$ is the state space of project i. The state of the system at time t is denoted as $\mathrm{s}(t) = (s_1(t), s_2(t), \ldots, s_N(t))$, with $s_i(t)$ the state of project i at time t.

- **Action space $\mathcal{A}$**: The set of available actions at each decision epoch, defined as the set of all possible combinations of M active projects out of the N available, $\binom{N}{M}$, with an action space defined as:

$$\mathcal{A} = \left\{ \mathbf{a} = (a_1, \ldots, a_N) | \sum_{i=1}^{N} a_i = M, a_i \in \{0, 1\} \right\} \tag{2.12}$$

  where $\mathbf{a}(t) = (a_1(t), a_2(t), \ldots, a_N(t))$ is the action vector at time t, with $a_i(t)$ the action for project i at time t. Under action $a_i = 1$, bandit i is active, while under $a_i = 0$, it remains passive. Under equation (2.12), only M projects can be active at each decision epoch while the remaining $N - M$ projects remain passive.

- **Transition probabilities $\mathbf{P}$**: Consider an action $\mathbf{a}$ taken at time t that satisfies equation (2.12). For $a_i(t) = 1$, the bandit i evolves according to its transition probability matrix:

$$P(s_i(t+1) = s' | s_i(t) = s, a_i(t) = 1) = P_i^1(s, s'), s, s' \in S_i,$$

  whereas for the passive bandits with $a_i(t) = 0$, it transitions with probability matrix:

$$P(s_i(t+1) = s | s_i(t) = s, a_i(t) = 0) = P_i^0(s, s'), s, s' \in S_i.$$

  The N bandits evolve independently, and the transition probability matrix for the whole system is given by:

$$\mathcal{P}^{\mathbf{a}}_{\mathbf{s},\mathbf{s}'} = \prod_{i=1}^{N} P_i^{a_i}(s_i, s_i'), \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S}, \mathbf{a} \in \mathcal{A}.$$

- **Reward function $\mathbf{R}$**: The reward function is defined as the expected reward received when transitioning from state s to state s' under action $\mathbf{a}$. It is bounded and stationary and defined as $R_i : S_i \times A_i \to \mathbb{R}$ for all i. For any transition from s to s' in bandit $i, 1 \leq i \leq N$, the discounted reward obtained is $\gamma^t R_i(s, a)$.

- **Number of projects** N: The number of MDPs or projects, each with its unique state space $\mathcal{S}_i$.
- **Number of active projects** M: The number of projects that can be active at each decision epoch.

The goal is to maximize the expected sum of discounted rewards over an infinite horizon, given the constraint that exactly M projects must be active at each decision epoch. We can, thus, define the optimal value function $V(\mathrm{s})$ for this process, evaluated at $\mathrm{s} \in \mathcal{S}$, as the maximal expected discounted reward from the initial state s, encapsulated by:

$$V(\mathbf{s}) = \max_{\mathbf{a} \in \mathcal{A}} \left(R(\mathbf{s}, \mathbf{a}) + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} \mathcal{P}^{\mathbf{a}}_{\mathbf{s},\mathbf{s}'} V(\mathbf{s}') | \sum_{i=1}^{N} a_i = M \right), \mathbf{s} \in \mathcal{S} \tag{2.13}$$

Considering a simplified scenario where $M = 1$ and passive actions do not alter the state nor yield rewards, we revert to the classical MAB framework from sections 2.2.1 and 2.2.2.

Building on Gittins [39] optimal policy for the Rested bandits, Whittle [42] proposed a heuristic based on indexability for RMAB problems, utilizing a Lagrangian relaxation to approximate the original problem's solution. In section 2.3 we will examine this Lagrangian relaxation in more detail and how it can evolve into heuristics for the Multi-Armed Bandits from sections 2.2.1 and 2.2.2, the Restless Multi-Armed Bandits and even Weakly Coupled MDPs from section 2.2.4.

Whittle's approach generalizes Gittins indices, adapting them to the restless context where both active and passive states evolve.

Indexability and the resulting Whittle indices are properties of the individual bandits. Hence, we will temporarily isolate an individual bandit and drop the bandit identifier i. In Whittle analysis, this bandit generates an MDP parametrized by a passive subsidy $W \in \mathbb{R}$. The W-subsidy problem bandit is defined by the tuple $\langle S, A, P, R \rangle$, where:

- **State space** S: The state space of the bandit.
- **Action space** A: The action space of the bandit, which can take either action $a = 1$ (active) or $a = 0$ (passive).
- **Transition probabilities** P: If action $a = 1$ is chosen at time t and the bandit is in state s, it evolves according to its transition probability matrix:

  $$P(s(t+1) = s' | s(t) = s, a(t) = 1) = P^1(s, s'), s, s' \in S,$$

  whereas for the passive bandit with $a = 0$, the transition probability matrix is:

  $$P(s(t+1) = s | s(t) = s, a(t) = 0) = P^0(s, s'), s, s' \in S.$$

We define P as the collection of passive and active transition matrices P^0 and P^1, $P = \{P^0, P^1\}$.

- **Reward function** R: The reward function is defined as the expected reward received when transitioning from state s to state s' under action a. Under action $a = 1$ at time t, the discounted reward $\gamma^t R(s, 1)$ is earned. On the other hand, under action $a = 0$, at time t the discounted reward becomes $\gamma^t (R(s, 0) + W)$, where W is the passive subsidy.

This W-subsidy problem redefines the value function $V(s, W)$ for each state s as:

$$\begin{aligned} V(s, W) = \max \Bigg\{ & R(s, 1) + \gamma \sum_{s' \in S} P^1(s, s') V(s', W); \\ & R(s, 0) + W + \gamma \sum_{s' \in S} P^0(s, s') V(s', W) \Bigg\} \end{aligned} \tag{2.14}$$

In this system, the active action will be optimal if the first term of the value function (2.14) is greater than the second term. Otherwise, the passive action will be the optimal one.

We define $\Pi(W)$ as the set of states where passive action is optimal under subsidy W. This set grows monotonically as W increases if the bandit is indexable.

Definition 2.7. *A bandit $\langle S, A, P, R \rangle$ is indexable if $\Pi(W)$ is increasing in W:*

$$W_1 < W_2 \implies \Pi(W_1) \subseteq \Pi(W_2).$$

A restless bandit is indexable when each of its individual bandits is indexable.

Indexability implies that a bandit's preference for passive action expands with the level of subsidy. Even though this might seem natural, this requirement is typically challenging to establish and sometimes fails to hold.

Definition 2.8. *If a bandit $\langle S, A, P, R \rangle$ is indexable, the Whittle index $W(s)$ for each state s is defined as the minimal subsidy level at which passivity becomes optimal:*

$$W(s) = \inf\{W : s \in \Pi(W)\}, s \in S.$$

Given the boundedness of the reward function, the Whittle index is guaranteed to exist and be finite for all states. The value of $W(s)$ reflects a *fair subsidy* that makes the passive action as attractive as the active one.

In practice, the Whittle index heuristic advocates for activating the M bandits with the highest current indices $W_i(s_i(t))$ at each time t, making the other $N - M$ bandits to passivity.

2.2.4 Weakly Coupled MDP with Continuous Action Spaces

The exploration of Weakly Coupled Markov Decision Processes (MDPs) involve an extension of Restless Multi-Armed Bandit Problems (RMABPs) binary decision problems, where the action space can take not only values outside the range $\mathcal{A} \in \{0,1\}^N$ but continuous values in an action space $\mathcal{A} \in [0, a_{\max}]^N$.

These problems are characterized by being composed of multiple sub Markov Decision Processes, each with independent state spaces, action spaces, and reward and transition probability functions. However, these processes are in turn coupled together by a constraint on the resource distribution: in each process i, with an action a_i and a state s_i, there is a cost associated with that action in that state $C(s_i, a_i)$ which is subtracted from the available resources B in that decision epoch. Formally, a Weakly Coupled MDP can be described as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathbf{P}, \mathbf{R}, \mathbf{C}, N, B \rangle$ where:

- **State space $\mathcal{S}$**: The combined state space of all projects, defined as the Cartesian product $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_N$, where $\mathcal{S}_i, 1 \leq i \leq N$ is the state space of project i. The state of the system at time t is denoted as $\mathbf{s}(t) = (s_1(t), s_2(t), \ldots, s_N(t))$, with $s_i(t)$ the state of project i at time t.

- **Activation cost $\mathbf{C}$**: The cost of acting on each project i at each state s_i with an action a_i. It is bounded and stationary and defined as $C_i : S_i \times A_i \rightarrow \mathbb{R}^+$ for all processes i.

- **Action space $\mathcal{A}$**: The set of available actions at each decision epoch, defined as the set of all possible combinations of actions that satisfy the resource constraint. The action space is defined as:

$$\mathcal{A} = \left\{ \mathbf{a} = (a_1, \ldots, a_N) | \sum_{i=1}^{N} C(s_i, a_i) \leq B, a_i \in [0, a_{\max}] \right\}, \tag{2.15}$$

  where $\mathbf{a}(t) = (a_1(t), a_2(t), \ldots, a_N(t))$ is the action vector at time t, with $a_i(t)$ the action for project i at time t. The resource constraint is defined as the sum of the costs of the actions taken at each decision epoch being less than or equal to the available resources B.

- **Transition probabilities $\mathbf{P}$**: Consider an action $\mathbf{a}$ taken at time t that satisfies equation (2.15). For each process i, with an action $a_i(t)$, its MDP evolves according to its transition probability matrix:

$$P(s_i(t+1) = s' | s_i(t) = s, a_i(t) = a) = P_i^a(s, s'), s, s' \in \mathcal{S}_i, a \in \mathcal{A}_i,$$

  The N MDPs evolve independently and the transition probability matrix for the whole system is given by:

$$\mathcal{P}^{\mathbf{a}}_{\mathbf{s},\mathbf{s}'} = \prod_{i=1}^{N} P_i^{a_i}(s_i, s_i'), \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S}, \mathbf{a} \in \mathcal{A}.$$

- **Reward function R**: The reward function is defined as the expected reward received when transitioning from state s to state s′ under action **a**. It is bounded and stationary and defined as $R_i : S_i \times A_i \to \mathbb{R}$ for all i. For any transition from s to s' in process i, $1 \leq i \leq N$, the discounted reward obtained is $\gamma^t R_i(s, a)$.
- **Number of projects** N: The number of MDPs or projects, each with its unique state space $\mathcal{S}_i$.
- **Available resources** B: The amount of resources available at each decision epoch. This variable is key in defining the action space $\mathcal{A}$ and the resource constraint defined in equation (2.15).

We can define the expected value function $V(\mathbf{s})$ for this process, evaluated at s $\in \mathcal{S}$, as the maximal expected discounted reward from the initial state s that follows the constraint action space defined in equation (2.15):

$$V(\mathbf{s}) = \max_{\mathbf{a} \in \mathcal{A}} \left(R(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}[V(\mathbf{s}') | \mathbf{s}, \mathbf{a}] \middle| \sum_{i=1}^{N} C(s_i, a_i) \leq B \right), \\ \mathbf{s} \in \mathcal{S}, a_i \in [0, a_{\max}], \forall i \in \{1, \ldots, N\} \tag{2.16}$$

Restless Multi-Armed Bandit Problems characterized by a binary action space in each of their MDPs, are therefore a particular case of Weakly Coupled MDPs, where the activation cost of each process is the binary action performed in that process, $C(s_i, a_i) = a_i, a_i \in \{0, 1\}, \forall i \in \{1, \ldots, N\}$, and the available resources B are the number of arms M to be activated in each decision epoch.

2.3 Lagrange relaxation

In the preceding sections, we explored the Multi-Armed Bandit problems (MAB), detailed in section 2.2.2, the Restless Multi-Armed Bandit Problems (RMAB) in section 2.2.3, and analyzed Weakly Coupled Markov Decision Processes (MDPs) with continuous action spaces as outlined in section 2.2.4. These frameworks collectively address scenarios involving multiple MDPs that are interconnected by constraints on the available action space. Specifically, the MAB framework restricts the activation to only one among N arms, whereas the RMAB scenario permits activating M out of N arms. The Weakly Coupled MDP framework extends these concepts further by introducing a generalized cost function $C(s_i, a_i)$ that operates over a set of resources B at each decision step.

A commonality among these models is the presence of N independent projects, each generating rewards and undergoing state transitions independently. This independence allows for the decomposition of the overarching problem into smaller, more manageable subproblems through the use of Lagrange multipliers. Consequently, a complex N-dimensional problem is transformed into N simpler one-dimensional subproblems.

In the following subsection, we aim to briefly present the seminal work of Hawkins [43], which shows the application of Lagrangian relaxation within Weakly Coupled MDPs and its subsequent applicability to MAB and RMAB contexts.

Consider the Weakly Coupled MDP represented as $\langle \mathcal{S}, \mathcal{A}, \mathbf{P}, \mathbf{R}, \mathbf{C}, N, B \rangle$ from section 2.2.4, along with its associated Bellman equation (2.16). The original formulation of this problem is generally considered intractable. To address these, we introduce a Lagrange Multiplier λ, applied uniformly across all states $\mathbf{s} \in \mathcal{S}$, which effectively incorporates the constraint directly into the optimization problem:

$$\tilde{V}(\mathbf{s}, \lambda) = \max_{\mathbf{a} \in \mathcal{A}} \left(R(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}[V(\mathbf{s}') | \mathbf{s}, \mathbf{a}] + \lambda (C(\mathbf{s}, \mathbf{a}) - B) \right), \tag{2.17}$$

Here, $\tilde{V}(\mathbf{s}, \lambda)$ approximates the true value function $V(\mathbf{s})$, denoted in equation (2.16). Subsequently, we define:

$$L(\mathbf{s}, \lambda) = \max \sum_{i=1}^{N} (R_i(s_i, a_i) - \lambda C(s_i, a_i)) + \gamma \mathbb{E}[L(\mathbf{s}', \lambda) | s_i, a_i] \tag{2.18}$$

This formulation maintains the transition probabilities and reward functions of the original value function (2.16) leading us to:

$$L(\mathbf{s}, \lambda) = \sum_{i=1}^{N} L_i(s_i, \lambda) + \lambda B \frac{1}{1 - \gamma}, \tag{2.19}$$

and,

$$L_i(s_i, \lambda) = \max_{a_i \in \mathcal{A}_i} \left\{ R_i(s_i, a_i) - \lambda C(s_i, a_i) + \gamma \mathbb{E}[L_i(s_i', \lambda) | s_i, a_i] \right\}. \tag{2.20}$$

This analysis reveals that $L(\mathbf{s}, \lambda)$ can be decomposed into N independent maximization tasks, each significantly smaller than the original problem, yet interconnected through the λ multiplier.

Given $L(\mathbf{s}) = \min_{\lambda \geq 0} L(\mathbf{s}, \lambda)$, we establish that:

- $L(\mathbf{s}) \geq V(\mathbf{s})$, a result derived from standard Lagrangian theory [44], underscoring foundational principles beyond the scope of this thesis.
- $L(\mathbf{s}, \lambda)$ is convex and piecewise linear with respect to λ.

These insights facilitate the formulation of a new problem formulation, wherein future constraints are relaxed, allowing decisions within each subproblem to solely influence the state and cost associated with that particular subproblem.

One notable challenge in solving equation (2.16) in its original form is estimating the term $\mathbb{E}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}]$. The framework of Lagrangian relaxation offers an estimation of this term:

$$\mathbb{E}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}] \leq \mathbb{E}[L(\mathbf{s}',\lambda)|\mathbf{s},\mathbf{a}] = \sum_{i=1}^{N} \mathbb{E}[L_i(s_i',\lambda)|s_i,a_i] + \lambda B \frac{1}{1-\gamma}, \tag{2.21}$$

where $L_i(s_i',\lambda)$ corresponds to equation (2.20). This leads us to define a new policy:

$$\mathbf{a}(\mathbf{s},\lambda) = \arg\max \left\{ \sum_{i=1}^{N} (R_i(s_i,a_i) + \gamma \mathbb{E}[L_i(s_i,\lambda)|s_i,a_i]) | \sum_{i=1}^{N} C(s_i,a_i) \leq B \right\}, \tag{2.22}$$

Where the term $\lambda B \frac{1}{1-\gamma}$ is omitted in this formulation as it does not influence the decision-making process $\mathbf{a}(\mathbf{s},\lambda)$.

In his exploration of Lagrangian relaxation, Hawkins [43] examines various methodologies for determining the optimal value of the Lagrange multiplier, λ, including setting it as $\lambda = 0$, which can be interpreted as ignoring future constraints, and setting λ as the $\arg\min$ of the Lagrange function of equation (2.19), as well as its computation through integer optimization and piecewise approximations. In this summary, we will focus on setting λ as the $\arg\min$ of the Lagrange function, which offers a structured approach to approximating the expected value function $\mathbb{E}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}]$.

A key step in this approach is to focus on computing $\mathbb{E}\left[\min_{\lambda \geq 0} L(\mathbf{s}',\lambda)|\mathbf{s},\mathbf{a}\right]$. This calculation is essential for approximating the expected value function, yet it is not decomposable as it depends on the state variable $\mathbf{s}'$, which in turn is dependent on decision $\mathbf{a}$. To decouple this constraint, we select a static λ, specifically, a λ that resolves the dual problem of (2.18). This is formalized as follows:

$$\begin{aligned} \min \quad & L(\mathbf{s},\lambda) \\ s.t. \quad & \lambda \geq 0 \end{aligned} \tag{2.23}$$

In this context, λ^* represents the minimizing Lagrange multiplier, serving as an approximate solution to $\mathbb{E}\left[\min_{\lambda \geq 0} L(\mathbf{s}',\lambda)|\mathbf{s},\mathbf{a}\right]$. This choice of λ^* is key in redefining the feasible policy from (2.22) as:

$$\mathbf{a}(\mathbf{s},\lambda^*) = \arg\max \left\{ \sum_{i=1}^{N} (R_i(s_i,a_i) + \gamma \mathbb{E}[L_i(s_i,\lambda^*)|s_i,a_i]) | \sum_{i=1}^{N} C(s_i,a_i) \leq B \right\}. \tag{2.24}$$

To further improve the decision-making framework, we introduce a decision set for each state and action pair as:

$$D_i(s_i, \lambda) = \arg\max_{a_i \in \mathcal{A}} L_i(s_i, \lambda). \tag{2.25}$$

Consequently, let $\mathbf{D}(\mathbf{s}, \lambda) = (D_1(s_1, \lambda), \ldots, D_N(s_N, \lambda))$ represent the comprehensive decision set across all MDPs. The ensuing policy involves solving the optimization problem:

$$\begin{aligned} \min \quad & \sum_{i=1}^{N} C(s_i, D_i(s_i, \lambda))\lambda \\ s.t. \quad & \sum_{i=1}^{N} C(s_i, D_i(s_i, \lambda)) \leq B \\ & \lambda \geq 0 \end{aligned} \tag{2.26}$$

And subsequently setting $\mathbf{a} = \mathbf{D}(\mathbf{s}, \lambda)$.

Looking ahead, we will now examine the setting of $C(s_i, a_i) = 1$ whenever $a_i = 1$. This leads us to the derivation of the Gittins index policy, which is optimal for the Multi-Armed Bandit Problem, and the Whittle index policy, which offers a near-optimal solution for the Restless Multi-Armed Bandit problem.

However, it is important to acknowledge that constraints within this framework can be nonlinear, complicating the minimization of $C(\mathbf{s}, \mathbf{a})\lambda$.

2.3.1 Lagrangian relaxation for Multi-Armed Bandit Problems

In the context of Multi-Armed Bandit (MAB) problems, the framework introduced in this section takes on a new form. A key consideration of the MAB problem is that the controller is permitted to select only one arm (or bandit) to activate at any given time, while the rest remain passive. We can adapt the MAB value function provided in (2.7) by redefining the constraint as $\sum_{i=1}^{N} C(s_i, a_i) = 1$. Here, the cost function is defined such that $C(s, a) = a$ for all states $s \in \mathcal{S}$ and actions $a \in \{0, 1\}$. In this way, in each decision epoch, the controller is only able to activate one arm, and the cost of activating that arm is 1. Equations (2.19) and (2.20) are then redefined as:

$$L(\mathbf{s}, \lambda) = \sum_{i=1}^{N} L_i(s_i, \lambda) + \frac{\lambda}{1-\gamma},$$

where

$$L_i(s_i, \lambda) = \max_{a_i \in \{0,1\}} \{R_i(s_i, a_i) - \lambda a_i + \gamma \mathbb{E}[L_i(s_i', \lambda) | s_i, a_i]\}.$$

A notable difference in this setting with respect to the general Weakly Coupled MDP is the definition of the constraint with an equality rather than an inequality. This alteration permits the Lagrange multiplier, λ, to assume negative values.

Consider the functions:

$$\begin{aligned} F_i^0(s_i, \lambda) &= R_i(s_i, 0) + \gamma \mathbb{E}[L_i(s_i, \lambda) | s_i, a_i = 0] \\ F_i^1(s_i, \lambda) &= R_i(s_i, 1) - \lambda + \gamma \mathbb{E}[L_i(s_i, \lambda) | s_i, a_i = 1] \end{aligned} \tag{2.27}$$

The decision set defined in (2.25) is redefined as:

$$D_i(s_i, \lambda) = \begin{cases} 1 & \text{if } F_i^1(s_i, \lambda) \geq F_i^0(s_i, \lambda) \\ 0 & \text{otherwise} \end{cases} \tag{2.28}$$

This implies that if the Lagrangian value function for the active action, as defined in equation (2.20), exceeds that of the passive action, then $D_i(s_i, \lambda)$ is set to 1, and 0 otherwise.

Given the Gittins index equation (2.8), Hawkins [43] proposed the following theorem:

Theorem 2.2. *For a given state* s*, let* $\lambda_{(1)} = \max_i \lambda_i(s_i)$ *represent the first order statistic, and* $\lambda_{(2)}$ *denote the second order statistic* $\lambda_{(2)} = \max_{i:\lambda_i(s_i)<\lambda_{(1)}} \lambda_i(s_i)$*. Assuming* $\lambda_{(2)} < \lambda_{(1)}$*, the following behavior is observed:*

$$\sum_{i=1}^{N} D_i(s_i, \lambda) \begin{cases} = 0 & \textit{for } \lambda > \lambda_{(1)} \\ = 1 & \textit{for } \lambda_{(2)} < \lambda \leq \lambda_{(1)} \\ \geq 2 & \textit{for } \lambda \leq \lambda_{(2)} \end{cases} \tag{2.29}$$

Furthermore, it is optimal for $\lambda_{(2)} < \lambda \leq \lambda_{(1)}$ *to set* $a_i = D_i(s_i, \lambda)$ *for* $i = 1, \ldots, N$*.*

This theorem indicates that within a specific range of λ values, it is optimal to set $\mathbf{u} = \mathbf{D}(\mathrm{s}, \lambda)$. This range aligns with the constraints defined the Gittins index policy, which was previously identified as optimal for the MAB problem (section 2.2.2).

Therefore, the overarching goal in this setting is to find a λ value that, when applied to (2.28) for all MDPs $i \in 1, \ldots, N$, results in the activation of a singular arm.

2.3.2 Lagrangian relaxation for Restless Multi-Armed Bandit Problems

In this section we will adapt the terms developed for the Lagrangian relaxation for multi-armed bandit from the section 2.3.1 for the Restless problem. Unlike the conventional Multi-Armed Bandit scenario, where the decision revolves around activating a single arm, the RMAB framework complicates the decision-making process by requiring the activation of M out of N available arms at any given time, as is shown in the constraints for the value function (2.13).

Given this new constraint, equations (2.19) and (2.20) become:

$$L(\mathbf{s}, \lambda) = \sum_{i=1}^{N} L_i(s_i, \lambda) + \lambda \frac{B}{1-\gamma}$$

where:

$$L_i(s_i, \lambda) = \max_{a_i \in \{0,1\}} \left\{ R_i(s_i, a_i) - \lambda a_i + \gamma \mathbb{E}[L_i(s_i', \lambda) | s_i, a_i] \right\}.$$

As in section 2.3.1, the use of an equality in the constraint instead of an inequality allows us to consider negative values for λ.

Given equation (2.27), the decision set becomes analogous to that defined in (2.28). The divergence in the RMAB scenario lies in selecting a specific value of λ that ensures exactly M bandits are activated. This can be represented by the requirement to find a λ for which the decision function $D_i(s_i, \lambda) = 1$ applies to precisely M bandits, that is, the selection of M arms based on the criterion that $F_i^1(s_i, \lambda) \geq F_i^0(s_i, \lambda)$ for those selected bandits.

2.4 Gittins and Whittle index computation

In the previous sections, we have introduced Lagrangian relaxation techniques applied to weakly coupled Markov decision processes (MDPs) and restless multi-armed bandit problems (RMABPs). These explorations highlighted how the relaxation methods simplify the complexity of managing multiple linked MDPs by decomposing them into more tractable subproblems. Building on this foundation, we now shift our focus to the numerical computation of Whittle indices for RMABPs. In particular, the algorithm presented in [45] provides a method for computing these indices in the context of RMABPs with binary action sets.

Given full knowledge of the transition probabilities, reward function, and cost function, which in the RMAB framework is $C(s_i, a_i) = a_i$, we can define the expected value and expected cost functions as:

$$f^{\pi} = \left(I - \gamma P(\mathbf{s}, \mathbf{a})\right)^{-1} R(\mathbf{s}, \mathbf{a}) \tag{2.30}$$

$$g^{\pi} = \left(I - \gamma P(\mathbf{s}, \mathbf{a})\right)^{-1} C(\mathbf{s}, \mathbf{a}) \tag{2.31}$$

Where γ denotes the discount factor relevant to discrete-time models, P represents the transition probability matrix, and R and C are the reward and cost functions, respectively, associated with a given vector of states and actions $\mathbf{s}, \mathbf{a}$. These quantities can be formally expressed as expected values as:

$$f^{\pi} = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} R(\mathbf{s}(t), \mathbf{a}(t))\right] \tag{2.32}$$

$$g^{\pi} = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} C(\mathbf{s}(t), \mathbf{a}(t))\right] \tag{2.33}$$

respectively, where $R(\mathbf{s}(t), \mathbf{a}(t))$ and $C(\mathbf{s}(t), \mathbf{a}(t))$ denote the reward and cost functions for the states and actions in a given time t. With these definitions, the optimization problem is defined as

$$\max_{\pi}[f^{\pi} - \lambda g^{\pi}]. \tag{2.34}$$

The core of the approach in [45] lies in splitting the state space into two distinct subsets: $\mathcal{S}_0$, the set of passive states, and $\mathcal{S}$, the full state space, where $\mathcal{S}_0 \subset \mathcal{S}$. If λ in (2.34) is too large, the objective prioritizes minimizing g^{π}, which effectively leads to the scenario where $\mathcal{S}_0 = \mathcal{S}$.

Niño-Mora presents the concept of the marginal work measure, denoted as

$$w_i^{\pi} = g_i^{\pi} - g_i^{\pi_0} = C(s_i, 1) - C(s_i, 0) + \gamma \sum_{j \in \mathcal{S}} \left(P_{ij}^{1} - P_{ij}^{0}\right) g_j^{\pi_0}. \tag{2.35}$$

This measure captures the incremental cost associated with transitioning from a passive to an active state and adopting the previous π_0 policy afterward. Similarly, the marginal reward measure is defined, expressing the difference in rewards by the expression

$$r_i^{\pi} = f_i^{\pi} - f_i^{\pi_0} = R(s_i, 1) - R(s_i, 0) + \gamma \sum_{j \in \mathcal{S}} \left(P_{ij}^{1} - P_{ij}^{0}\right) f_j^{\pi_0}. \tag{2.36}$$

The decision criterion revolves around the comparison of marginal productivity measures,

$$\lambda_i^{\pi} = \frac{r_i^{\pi}}{w_i^{\pi}}, \tag{2.37}$$

when $w_i^{\pi} \neq 0$. This ratio provides a quantitative evaluation of the efficiency of activating an arm, balancing the trade-off between additional rewards and the costs incurred.

In an extension of the work initiated by [45], [46] broadens the scope to accommodate multi-action scenarios, incorporating a more diverse set of actions $a = \{0, 1, 2, \ldots, a_M\}$.

Weber's methodology divides the state space into a series of subsets $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{M-1}$ along with $\mathcal{S}$. Within this structure, for each state s spanning the subsets $\{\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{M-1}\}$, Weber formulates the index

$$\lambda_i^{\mathcal{S}_j, \mathcal{S}_{j+1}} = \frac{f_i^{\langle j_i+1, \mathcal{S}_j \rangle} - f_i^{\langle j_i, \mathcal{S}_j \rangle}}{g_i^{\langle j_i+1, \mathcal{S}_j \rangle} - g_i^{\langle j_i, \mathcal{S}_j \rangle}}, \tag{2.38}$$

where j_i denotes the action taken in state i.

This formulation defines the marginal productivity associated with increasing the action $a = j_i$ by one unit for state i, in the context of transitioning from policy $\mathcal{S}_j$ and maintaining that policy.

With an action set encompassing a_M possible actions over $|\mathcal{S}|$ distinct states, the resulting framework requires the computation of $(a_M - 1)|\mathcal{S}|$ indices.

Weber's approach involves a systematic progression of each state through the set of subsets, starting at $\mathcal{S}_0$.

Chapter 3

Reinforcement Learning Algorithms

In the previous chapter, we explored the basic frameworks of Markov decision processes (MDPs), restless multi-armed bandit (RMAB) problems and weakly coupled MDPs. These models are essential for formalizing decision processes in which actions taken in uncertain environments affect future states and associated rewards. However, a common challenge in these scenarios is the lack of complete information about the system dynamics. This information is often either partially known or completely unknown. In such cases, reinforcement learning (RL) techniques provide a powerful framework for learning optimal policies through interaction with the environment.

Reinforcement Learning (RL) has emerged as a powerful set of techniques to address these challenges. RL does not require explicit knowledge of the underlying model; instead, it focuses on learning optimal policies through trial and error, interacting directly with the environment to maximize cumulative rewards. This approach is not only applicable to RMABPs and weakly coupled MDPs, but also extends to a wide range of problems in diverse domains, including robotics, automated control, and economics, where decision agents must learn to make sequences of decisions under uncertainty.

This chapter introduces some RL techniques, such as tabular Q-learning and SARSA (section 3.1), which are fundamental for dealing with environments where the state and action spaces are sufficiently small to allow for a table-based approach. In addition, we discuss how function approximators (section 3.2), specially neural networks (section 3.2.1), are integrated into RL, providing the means to handle larger, high-dimensional state spaces by approximating the value functions and policies needed to effectively navigate more complex decision environments. Using these techniques, we can develop robust policies that maximize rewards.

3.1 Tabular Q-learning and SARSA

The Q-learning algorithm is a fundamental reinforcement learning technique. It serves as the basis for the computation of the Q-values introduced in equations (2.3) and (2.5), which represent the expected utility of taking a specific action in a given state and following a given policy thereafter.

Q-learning, introduced in [17], aims to determine the Q-values associated with state-action pairs by iteratively updating these values based on the observed transitions. Within each decision epoch t, the agent observes a transition $(s(t), a(t), r(t), s(t+1))$ that encapsulates the current state $s(t)$, the action taken $a(t)$, the reward received $r(t)$, and the following state $s(t+1)$.

Q-learning is a model-free and value-based algorithm, which enables learning the value functions using the observed transitions without prior knowledge of the environment's transition and reward dynamics. Model-based algorithms, in contrast, require a priori knowledge of the environment's dynamics, whereas policy-based algorithms directly learn the optimal policy without estimating the value function.

It is characterized as well by its off-policy nature, which means that it evaluates and improves a policy that is different from the one used during the data collection process. Being off-policy allows the algorithm to learn from a wider range of experiences beyond the actions taken by the current policy, by decoupling the exploration and exploitation strategies. The algorithm's decoupling is evident in several key aspects: First, Q-learning typically employs an exploration-focused strategy, such as the ϵ greedy policy, during the training phase. This strategy dictates that the agent chooses the action with the highest Q-value with probability $1-\epsilon$ and a random action with probability ϵ. This approach facilitates the exploration of the action space and ensures that the agent does not prematurely converge to a suboptimal policy by always exploiting the currently known best actions. Second, the computation of expected future rewards in Q-learning explicitly reflects its off-policy nature: Q-learning estimates future rewards based on the assumption that the optimal action will be taken thereafter.

In tabular Q-learning, the agent initializes a Q-table, a comprehensive matrix correlating every possible state-action pair with a value. This table is the agent's evolving knowledge base, guiding decision-making processes as it learns.

Initially, a higher ϵ encourages exploration to gain a broad understanding of the environment. As learning progresses, ϵ is gradually reduced to focus more on exploiting known information to optimize actions according to the Q-table.

When an action is executed, the environment transitions and provides new data $(s(t), a(t), r(t), s(t+1))$. The Q-table is updated with this new information, refining the agent's understanding and strategy. The update mechanism uses temporal difference error and incorporates immediate rewards, discounted future rewards, and existing Q-values to adjust estimates:

$$
\begin{aligned}
Q_{n+1}(s(t), a(t)) \leftarrow & Q_n(s(t), a(t)) + \\
& \alpha(t)\left(r(t) + \gamma \max_v Q(s(t+1), v) - Q(s(t), a(t))\right),
\end{aligned} \tag{3.1}
$$

where α is the learning rate, that needs to satisfy $\sum_{t=0}^{\infty} \alpha(t) \rightarrow \infty$, $\sum_{t=0}^{\infty} \alpha^2(t) < \infty$, γ is the discount factor defined in $0 \leq \gamma < 1$, and v denotes the possible actions in the subsequent state $s(t+1)$.

This method differs from the SARSA ("State-Action-Reward-State-Action") algorithm, introduced as a technical note in [35] and renamed by Richard Sutton, which is an on-policy algorithm. In SARSA, the policy used in training is the final policy, which not only includes the exploration method used, but also affects the future rewards used to calculate these Q-values:

$$
\begin{aligned}
Q_{n+1}(s(t), a(t)) \leftarrow & Q_n(s(t), a(t)) + \\
& \alpha(t)\left(r(t) + \gamma Q(s(t+1), a(t+1)) - Q(s(t), a(t))\right),
\end{aligned} \tag{3.2}
$$

where the future expected rewards $Q(s(t+1), a(t+1))$ (marked in red in Equation (3.2)) use the current policy future actions $a(t+1)$, which might lead to a suboptimal action given an exploration strategy.

3.2 Function Approximation

In tabular frameworks such as Q-learning and SARSA, each state and action in the environment must be visited frequently enough to accurately estimate the value functions. This requirement can become prohibitively expensive in environments with large or continuous state spaces, where the number of possible states and actions can be large or infinite. As a result, achieving sufficient sampling of all state-action pairs to learn effective policies becomes computationally infeasible.

Function approximators provide a powerful solution to this challenge by providing generalization across similar states. Rather than storing and updating a value for each individual state-action pair, function approximators allow the agent to infer values for unvisited states based on the properties and outcomes of states that have been explored. This capability not only reduces the dimensionality and complexity of the learning problem, but also increases the efficiency and scalability of reinforcement learning algorithms. Using techniques such as neural networks, linear regression, or decision trees, these approximators can effectively capture the essential characteristics of the environment's dynamics, enabling robust policy learning even in complex scenarios with large or unbounded state spaces.

There are many models of function approximation, all of which are suitable for different types of problems and applications. Linear models, for example, assume a linear relationship between input features and target outputs. They provide a level of simplicity and ease of interpretation that is valuable in scenarios where such a relationship is the dominant one. Neural networks, on the other hand, are extremely well suited for dealing with more dynamic environments, thanks to their remarkable flexibility and ability to capture complex, nonlinear relationships. Decision trees provide an intuitive, hierarchical way to navigate through choices by organizing decisions and their potential outcomes into a tree. Building on this concept, random forests extend the decision tree model by merging multiple trees to increase the accuracy of the prediction and reduce the risk of over-fitting through the principles of ensemble learning. Kernel methods, including support vector machines (SVMs) with nonlinear kernels, excel at implicitly mapping input features into higher-dimensional spaces. This allows modeling of nonlinear relationships without direct computation in these expanded dimensions. Gaussian processes excel at not only providing predictions, but also assessing the uncertainty surrounding those predictions by introducing a probabilistic lens to function approximation. This feature is particularly valuable for making informed decisions under conditions of uncertainty.

Within the reinforcement learning context, function approximators are instrumental across several domains:

- **Value Function Approximation** aims at estimating the value of states or state/action pairs to guide decision making.
- **Policy Approximation** involves the direct estimation of policies, making a probabilistic mapping of actions given a state.
- **Model Approximation** focuses on capturing the dynamics of the environment, predicting subsequent states and rewards from current states and actions.

3.2.1 Neural Networks and its application in RL

Neural networks, first introduced by [47], represent a paradigm shift in computational approaches. They are inspired by the biological neural networks observed in animal brains. These models excel at pattern recognition and decision-making, especially in tasks involving complex input-output relationships. Their ability to learn from data allows them to tackle a wide range of challenges that are intractable by traditional computational methods.

The architecture of a neural network consists of a series of interconnected nodes, called artificial neurons or units. These units are systematically organized into distinct layers:

- At the core of these layers are the **neurons**, the basic processing units of the network. Each neuron computes a weighted sum of its inputs, adds a bias, and then passes this sum through an activation function to produce an output.

- **Weights and biases** are the learnable parameters of the network. Weights adjust the strength of connections between neurons, while biases shift the activation function.
- The **activation function** introduces nonlinearity into the network, allowing it to capture complex patterns. Commonly used activation functions include sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax, with ReLU being the most widely used.
- The **input layer** receives the raw input data, with each neuron within this layer representing a specific feature of the input.
- **Hidden Layers** lie between the input and output layers and perform most of the computational processing and feature transformation. A neural network can have one or more hidden layers, each containing any number of neurons.
- The **output layer** provides the final output of the network. The configuration of this layer is tailored to the task at hand, whether it is classification, regression, or some other form of decision making.

Neural networks work through a process known as forward propagation. In this process, input data passes through the network from the input layer to the output layer. At each neuron, the inputs from the previous layer (or the original input data for the first layer) are multiplied by the neuron's weights, summed, and then adjusted by a bias term.

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

for layer l, where $a^{(l-1)}$ is the input vector to layer l with dimension $(n_{l-1} \times 1)$, and $W^{(l)}$ and $b^{(l)}$ are the weights and bias, being a matrix with size $(n_l \times n_{l-1})$ and a vector of size $(n_l \times 1)$ respectively. The result of this linear transformation, $z^{(l)}$, is then passed through an activation function $g(\cdot)$ applied element-wise to the vector $\mathbf{z}^{(l)}$, leading to the output

$$\mathbf{a}^{(l)} = g(\mathbf{z}^{(l)})$$

The loss function quantifies the discrepancy between the network's predictions and the actual target values. Common choices include mean squared error (MSE) for regression and cross-entropy for classification. The equation for the MSE is

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where L is the loss value, N is the number of samples, y_i is the actual target value, and $\hat{y}_i$ is the predicted value. The equation for the cross-entropy loss is

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

where y_i is the actual target value (0 or 1), and $\hat{y}_i$ is the predicted probability of the positive class.

Backpropagation, introduced in [48], is the mechanism by which the network updates its parameters. It involves calculating the gradient of the loss function with respect to each parameter (weights and biases), and propagating these gradients back through the network to inform updates. This is accomplished by applying the chain rule of calculus, starting with the output layer and working backwards.

1. **Initialization of Gradient Calculation at the Output Layer**: The process begins by calculating the gradient of the loss function with respect to the activations of the output layer. This is mathematically represented as:

   $$\delta^{(L)} = \frac{\partial L}{\partial a^{(L)}} g'(z^{(L)})$$

   where $\delta^{(L)}$ denotes the gradient of the loss with respect to the output activation $a^{(L)}$ at layer L, and $g'(\cdot)$ is the derivative of the activation function.

2. **Backward Propagation of the Gradient**: With the gradient at the output layer computed, the next step involves propagating this gradient backward through the network to compute the gradients with respect to the activations of each preceding layer. For each layer $l = L-1, L-2, \ldots, 1$, the gradient $\delta^{(l)}$ is calculated using the gradient from the subsequent layer $(l+1)$ as:

   $$\delta^{(l)} = ((W^{(l+1)})^T \delta^{(l+1)}) g'(z^{(l)}).$$

   This equation determines the gradient at layer l by considering the effect of the gradient at layer $(l+1)$ and the derivative of the activation function at layer l.

3. **Calculation of the Gradients with Respect to Weights and Biases**: With the gradients of the loss with respect to the activations computed, the next step involves determining how the loss changes with respect to the weights and biases. The gradients are given by:

   $$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)} (a^{(l-1)})^T$$

   and

   $$\frac{\partial L}{\partial b^{(l)}} = \delta^{(l)}$$

   These equations compute the gradients of the loss function with respect to the weights and biases at layer l.

4. **Update of Weights and Biases using Gradient Descent**: The final step involves updating the weights and biases using the gradients computed in the previous step. This is achieved through an iterative optimization algorithm,

typically gradient descent or one of its variants, which incrementally adjusts the parameters to minimize the loss. The update equations are given by:

$$W^{(l)} = W^{(l)} - \alpha \frac{\partial L}{\partial W^{(l)}}$$

and

$$b^{(l)} = b^{(l)} - \alpha \frac{\partial L}{\partial b^{(l)}}$$

where α is the learning rate, a hyperparameter that controls the size of the update steps.

More recently, the Adam optimizer [49] has gained popularity for its adaptive learning rate mechanism, which adjusts the learning rate for each parameter based on the first and second moments of the gradients. This optimizer is known for its effectiveness in adjusting neural network parameters through an adaptive learning rate mechanism. The process is performed in the following steps:

1. **Gradient Computation**: First, the gradient of the loss function with respect to the neural network parameters is computed. This gradient, denoted as $g_t = \nabla_\theta L(\theta)$, for each parameter θ, provides the direction in which the parameters should be adjusted to minimize the loss.

2. **Moment Updating**: Adam distinguishes itself by tracking two moments for each parameter, the first moment (mean) m_t and the second moment (uncentered variance) v_t. These moments are updated in the following way:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t, \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \end{aligned} \tag{3.3}$$

   where β_1 and β_2 are hyperparameters that control the exponential decay rates of these moving averages, generally set close to 1. This dual-moment mechanism allows Adam to adaptively adjust learning rates based on both the first and second order moments of the gradients.

3. **Bias Correction**: One of Adam's features is its implementation of bias correction for both moments. This step is critical in the early stages of training when the moments may be biased towards zero, potentially slowing down the learning process. The corrected moments are calculated as

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t}, \end{aligned} \tag{3.4}$$

   where t represents the current timestep. These bias-corrected moments $\hat{m}_t$ and $\hat{v}_t$ provide more accurate estimates for adjusting the parameters.

4. **Parameter Update**: Once the gradients and corrected moments are computed, the parameters are updated using an adaptive learning rate specific to each parameter. This update is governed by the equation:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t, \tag{3.5}$$

   where η signifies the learning rate and ϵ is a small scalar (e.g., 10^{-8}) added to enhance numerical stability. This update rule ensures that each parameter is adjusted based on its own historical gradient information.

There are a variety of architectures and types of neural networks, each for different types of problems. Among them, the following types stand out:

- **Feedforward Neural Networks (FNN)** represent the simplest form of neural networks [50], where connections between the nodes do not form cycles. This straightforward architecture allows for a clear, directed flow of information from input to output layers, making FNNs particularly suited for a wide range of prediction and classification tasks.

- **Convolutional Neural Networks (CNN)**, introduced in [51], have proven to be extremely effective for tasks involving image data. Using convolutional layers, CNNs can capture spatial features, making them indispensable in areas such as image recognition and computer vision.

- **Recurrent Neural Networks (RNN)** are designed to handle sequential data, such as time series, speech, or text. They were first introduced [52], and their unique architecture, which allows connections to form cycles, enables RNNs to maintain a form of memory of previous inputs. This feature makes them well suited for applications that require an understanding of temporal dynamics.

- **Deep Neural Networks (DNN)** are characterized by their multiple hidden layers, which enable them to learn complex patterns within the data. The term was popularized by [53] and [54]. The depth of these networks is a key factor in their ability to model highly complex relationships.

- **Generative Adversarial Networks (GAN)**, first introduced in [55], consist of two competing networks: a generator that creates data instances and a discriminator that evaluates them. This setup allows GANs to generate new data instances that mimic the training data, finding applications in areas such as image generation and style transfer.

Deep Q-Networks (DQN) represent a significant advance in reinforcement learning, combining the classical tabular Q-learning algorithm with the power of deep neural networks. Introduced by [20], DQNs achieved remarkable success in mastering Atari 2600 games, often outperforming human players. By processing the state of the environment as input and outputting the predicted Q-values for each possible action, DQNs effectively manage high-dimensional state spaces.

These algorithms use several key components, which together contribute to the robustness and effectiveness of DQNs in learning optimal policies. The experience replay is fundamental, using a replay buffer to record the experiences of the agent, represented as tuples of $\langle s, a, r, s' \rangle$. This method allows training on random batches of past experiences, significantly improving learning efficiency and stability by breaking the temporal correlation of successive samples.

Another key feature of the DQN architecture is the implementation of Q-targets, which involves the use of a secondary network specifically tasked with generating Q-values for updating targets within the Bellman equation. The purpose of these networks is critical for stabilizing the learning dynamics, as it mitigates the risk of rapid shifts in Q-value estimates that could potentially derail the learning process. The weights of this target network are periodically updated to reflect those of the main network, either by replacing the parameters of the secondary network with those from the main network, or using a soft update formula: $\theta_{target} \leftarrow \tau\theta_{main} + (1 - \tau)\theta_{target}$, ensuring a gradual and controlled update.

Finally, the choice of the loss function in DQNs, typically the Mean Squared Error (MSE), is critical for quantifying learning performance. The MSE calculates the mean squared difference between the predicted Q-values and the target Q-values, $MSE = \frac{1}{N}\sum(Q(s, a) - Q'(s, a))^2$, providing a clear metric of the accuracy of the network's performance relative to the expected results. This loss function is critical in guiding the optimization process, allowing prediction errors to be systematically reduced through successive training iterations.

To further improve the performance of the basic Deep Q-Network (DQN) framework, several modifications have been introduced. Prioritized Experience Replay [56] improves learning efficiency by giving higher priority to more informative experiences within the replay buffer, particularly emphasizing rare or underused samples during training. In addition, Dueling DQN [57] refines the architecture by making a clear distinction between the values of states and the benefits conferred by each action. This separation enhances the algorithm's ability to learn effectively, especially in scenarios where the choice between actions does not significantly change the outcome.

Chapter 4

Discrete Action Models in RMABP

This chapter focuses on the development and analysis of algorithms for the Restless Multi-Armed Bandit (RMAB) problem, a complex extension of the traditional multi-armed bandit framework. Consider the RMAB problem described in Section 2.2.3, denoted as $\langle \mathcal{S}, \mathcal{A}, \mathbf{P}, \mathbf{R}, N, M \rangle$, which extends the MAB problem where each of the N arms, indexed by $i = 1, \ldots, N$, has its own state space $\mathcal{S}_i$ with size $|\mathcal{S}_i|$. The coupled state space $\mathcal{S}$ is defined as $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_N$ with size $|\mathcal{S}|$. At each time step t, an arm is in state $s_i(t)$, upon which an action $a_i(t) \in \{0, 1\}$ is taken, resulting in a reward $R_i(s_i(t), a_i(t))$. The transition probabilities for active and passive arms are given by $P^1(s(t+1), s(t))$ and $P^0(s(t+1), s(t))$, respectively. From the N available arms, only M can be set with $a_i(t) = 1$, while the remaining $M - N$ arms are kept passive with $a_i(t) = 0$.

Our objective is to identify a control policy π that activates those M arms at each decision epoch to maximize the expected cumulative discounted reward, formalized as:

$$\begin{aligned} V(\mathbf{s}) = \max_{\pi} \mathbb{E} \left[\sum_{t=0}^{\infty} \sum_{i=1}^{N} \gamma^t R(s_i(t), a_i(t)) \right] \\ \text{s.t.} \sum_{i=1}^{N} a_i = M, \\ s_i(t) \in \mathcal{S}_i, a_i \in \mathcal{A}_i = \{0, 1\}, \text{ and } t \geq 0, \end{aligned} \tag{4.1}$$

as discussed in Section 2.2.3, using a discount factor γ where γ is de discount factor defined in $0 < \gamma < 1$ (see Section 2.1.2).

To approach this problem, we build on Whittle's insight [42], which suggests a relaxation of the problem through Lagrangian relaxation, leading to a modified objective:

$$V'(s_1, \ldots, s_N) = \max_{\pi'} \mathbb{E} \left[\sum_{t=0}^{\infty} \sum_{i=1}^{N} \gamma \left(R_i(s_i(t), a_i(t)) + \lambda(1 - a_i(t)) \right) \right] \tag{4.2}$$

This relaxation introduces a subsidy for passivity λ, implicitly encouraging the selection of the passive action in more states as λ increases.

The key of solving the RMAB problem lies then in decomposing it into individual problems for each arm i, using the associated Bellman equation:

$$V_i(s_i) = \max_{a \in \{0,1\}} Q_i(s_i, a) \tag{4.3}$$

where $Q_i(s_i, a)$ is defined as:

$$\begin{aligned} Q_i(s_i, a) = a \left(R_i(s_i, 1) + \gamma \sum_{j \in \mathcal{S}_i} P_i^1(j, s) V_i(j) \right) + \\ (1 - a) \left(R_i(s_i, 0) + \lambda + \gamma \sum_{j \in \mathcal{S}_i} P_i^0(j, s) V_i(j) \right) \end{aligned} \tag{4.4}$$

The Whittle index for a state s is defined for the choice between activating and not activating an arm is indifferent, i.e.,

$$\lambda(s) : Q(s, 1) - Q(s, 0) = 0. \tag{4.5}$$

One of the properties of the RMAB problem is its generalization of the MAB problem: when $M = 1$, and the reward and transition probabilities for the passive state are null, the RMAB problem reduces to the classical MAB problem. Furthermore, the Whittle index generalizes to the Gittins index for this case, which is a well-known optimal heuristic for the MAB problem [39].

This introduction provides the framework for the presentation of two discrete-action algorithms developed during this thesis: the QWI (*Q-Learning for Whittle Index*) and QWINN (*Q-Learning for Whittle Index Neural Network*) algorithms, which aim to efficiently solve the RMAB problem. In the following sections, we will discuss these algorithms in detail, including their theoretical foundations, algorithm design, and performance.

4.1 QWI: Tabular Learning of the Whittle Indices

The computation of Q-values (equation (4.4)) and Whittle indices (equation (4.5)) share a common dependency: the accurate estimation of Q-values is essential for the calculation of Whittle indices, but these indices in turn influence the Q-values. This dependency creates a cyclic link in which adjusting one parameter affects the other, leading to instability in the learning process.

Traditional approaches to this dilemma have typically adopted a sequential update mechanism, dedicating extended periods to refining the Q-values before proceeding to adjust the Whittle indices based on these updated values [58, 59].

Appendix A.1 provides the theoretical foundation for the QWI algorithm, which is based on the two-timescale stochastic approximation method. This method allows for the simultaneous updating of the Q-values and Whittle indices, ensuring the convergence of both parameters. In this Section, we will follow Borkar's [60] analysis of multiple timescales in Markov Decision Processes (MDPs), which provides a robust framework for simultaneously updating interdependent parameters. Borkar's work lays the groundwork for a dual timescale stochastic approximation method in which the Q-values and Whittle indices are updated simultaneously but at different rates: a fast timescale for the Q-values and a slow timescale for the Whittle indices.

In the following Section, we will present the QWI [29] algorithm design and implementation, which leverages the two-timescale stochastic approximation method to efficiently learn the Whittle indices for each arm in the RMAB problem.

4.1.1 Algorithm Design and Implementation

The Q-learning algorithm, introduced in Section 3.1, serves as the foundation for the QWI algorithm. Building on the basic concepts of tabular Q-learning, the QWI algorithm is designed to address the challenges posed by the RMAB problem. The primary goal of the algorithm is to learn the Whittle indices for each arm by decomposing the RMAB problem into its various subproblems, which are then used to guide the decision process.

The core idea behind QWI lies in iteratively adjusting both the Whittle index and the state-action values until convergence occurs. The Whittle index $\lambda(x)$ for a state $x \in \mathcal{S}_i$ can be implicitly derived from the equilibrium condition in equation (4.5). This is an implicit equation since the state-action function $Q(\cdot, \cdot)$ depends on the value of the multiplier $\lambda(x)$. The Whittle index is thus defined by the coupled equations (4.4) and (4.5).

We will update Equation (4.4) to reflect the computation of the Q-values using the specific Whittle index $\lambda(x)$ instead of λ. For each arm i and each state $x \in \mathcal{S}_i$:

$$\begin{aligned} Q^x_{n+1}(s(n), a(n)) =& (1-\alpha(n))Q^x_n(s(n), a(n)) + \\ & \alpha(n)\left((1-a(n))(R_0(s(n)) + \lambda_n(x)) + \right. \\ & \left. a(n)R_1(s(n)) + \gamma \max_{v \in \{0,1\}} Q^x_n(s(n+1), v)\right) \end{aligned} \tag{4.6}$$

and

$$\lambda_{n+1}(x) = \lambda_n(x) + \beta(n)\left(Q^x_n(x,1) - Q^x_n(x,0)\right), \tag{4.7}$$

where $s(n)$ is the state visited at time step n, x is a reference state for which we want to learn the Whittle index, $R_0(s(n))$ and $R_1(s(n))$ are the sampled rewards for passive and active actions, respectively, and $\alpha(n)$ and $\beta(n)$ are learning parameters. The superscripts x in (4.6) stand for the parametric dependence of the Q-values on $x, \lambda(x)$, where x is a fixed reference state and $\lambda(x)$ is a slowly varying parameter updated by (4.7). As explained in the Section A.1, the learning rates $\alpha(n)$ and $\beta(n)$ are chosen to satisfy the conditions for the two-timescale stochastic approximation method, ensuring convergence of the Q-values and Whittle indices:

$$\sum_n \alpha(n) = \infty, \sum_n \alpha(n)^2 < \infty, \sum_n \beta(n) = \infty, \sum_n \beta(n)^2 < \infty \text{ and } \beta(n) = o(\alpha(n)).$$

In our case, we use the following step sizes that meet the above-mentioned conditions:

$$\alpha(n) = \frac{1}{\lceil n/5000 \rceil} \tag{4.8}$$

$$\beta(n) = \frac{1}{1 + \lceil n \log(n)/5000 \rceil} I\{(n) \bmod(50) = 0\} \tag{4.9}$$

where $\beta(n)$ is non-zero only once each 50 iterations.

Theorem 4.1. *The QWI algorithm's Whittle index estimates converge to the optimal Whittle indices.*

Proof: In Appendix A.2 we provide a proof of the convergence of the Whittle index estimates to the optimal Whittle indices. The proof is based on the two-timescale stochastic approximation method discussed in Section A.1.

The pseudocode implementation of QWI can be found in Algorithm 1. For a given value of the discount parameter $\gamma < 1$ and the exploration parameter $0 \leq \epsilon \leq 1$, we initialize the N arms with random states. At each decision epoch, denoted by n, actions are determined based on a balance between exploration and exploitation: with a probability of $1 - \epsilon$, the algorithm chooses a greedy strategy, activating the M arms with the highest $\lambda_n(x)$ values; on the other hand, with a probability of ϵ, a random selection of M arms is made. Once the action has been executed, the rewards $R^i(n)$ are collected and the resulting states $s^i(n+1)$ are observed for each arm. The updates to both the state-action value functions and the Whittle indices are then updated according to the equations (4.6) and (4.7).

The update process for a given arm i involves a comprehensive iteration over all possible states within its state space $\mathcal{S}_i$, using each reference state x. This iterative nature implies that the computational complexity of the QWI algorithm - quantified as the total number of computations required for these updates - is proportional to $O(|\mathcal{S}_i|^2, |\mathcal{A}|)$, where the action space $\mathcal{A}$ encompasses the binary decisions $0, 1$. This method is significantly more computationally efficient than the original tabular Q-learning application discussed in (3.1), where the MDPs are coupled and therefore their state space size is $|\mathcal{S}| = |\mathcal{S}_i|^N$, leading to complexity $O(|\mathcal{S}_i|^N, |\mathcal{A}|)$.

For the implementation of the QWI algorithm within this thesis, Python 3.10 serves as the primary programming language. Python offers versatility and extensive support for scientific computing, making it an ideal choice for developing and testing reinforcement learning algorithms.

Algorithm 1 Tabular QWI Algorithm

```
Input: Discount parameter $\gamma \in (0, 1)$, exploration parameter $\epsilon \in [0, 1]$,
Output: Whittle index matrix for all states in each arm $i$
Initialize $s_0$ for all arms
for $n = 1 : n_{end}$ do
    Define action $a_i(n)$ through $\epsilon$-greedy policy for each arm $i$
    Get new states $s_i(n + 1)$ and rewards $r_i(n)$ from states $s_i(n)$ and actions $a_i(n)$

    Update learning rate $\alpha(n)$, $\beta(n)$ as (4.8) and (4.9)
    for $x \in \mathcal{S}_i$ do
        Update $\langle s_i(n), a_i(n), x\rangle$ $Q$-values as (4.6) $\forall i \in [1, N]$
        Update Whittle estimate for state $x$ in each arm $i$ as (4.7)
    end for
end for
```

The only external library used in the implementation of the QWI algorithm is Numpy, a library known for its powerful array manipulation and mathematical operations. Its efficiency in handling large datasets and performing complex numerical computations is crucial in facilitating the algorithm's processing needs, allowing for effective and efficient experimentation within the RMAB problem space.

4.2 QWINN: Enhancing QWI with Neural Networks

Traditional approaches to the RMAB problems require exhaustive exploration of each state and action to converge to optimal policies. Due to this, they often fail due to the curse of dimensionality. In particular, in RMAB scenarios, the computational complexity escalates as $O(|\mathcal{S}_i|^N, |\mathcal{A}|)$, where N is the number of arms involved.

The QWI algorithm represents a significant advance by reducing this complexity to $O(|\mathcal{S}_i|^2, |\mathcal{A}|)$, achieved by decoupling the arms via the Whittle indices. However, this approach does not completely circumvent the challenge of comprehensive state space exploration for individual arms. The requirement to explore the entirety of a single arm's state space-especially when faced with states that are infrequently visited or exhibit significant variability in outcomes-can significantly slow the convergence of both Q-values and, consequently, the Whittle indices.

This limitation is not unique to QWI, but is a challenge that is characteristic of tabular methods, where the data for each pair of states and actions is isolated, preventing the exploitation of insights across similar states or actions. To address these challenges, function approximators, previously introduced in Section 3.2, have emerged as powerful tools in machine learning and reinforcement learning, among other fields, providing a means to generalize from observed data and predict outcomes for unseen data points.

The development of QWINN [29], an innovative hybrid algorithm, utilizes the integration of Feedforward Neural Networks (FNN) to compute Q-values within the system, as introduced in Section 3.2.1, while maintaining a tabular approach for the computation of Whittle indices. This combination leverages the strengths of neural networks for handling the complex relationships inherent in the state-action spaces of RMAB problems and the proven effectiveness of tabular methods for index computation.

The neural network used in QWINN is structured with three hidden layers of 100, 200, and 100 neurons, respectively. Each layer is connected by ReLU activation functions, which facilitates the network's ability to effectively capture and model nonlinear dynamics while being highly optimized. The input to the network is structured to incorporate two state variables: the visited state s and the Whittle index reference state x, which refers to $\lambda(x)$. Consequently, the outputs of this neural network are the Q-values for the two possible actions, represented as $Q^x_\theta(s) = [Q^x_\theta(s,0) \quad Q^x_\theta(s,1)]$.

The training process of QWINN proceeds iteratively, where at each step a set of actions $\mathbf{a}$ is chosen based on the Whittle Index Heuristic, given the current states $\mathbf{s}$. In scenarios where all arms have identical properties, the transition tuple $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}' \rangle$ for each arm i is decomposed into individual tuples $\langle s_i, a_i, r_i, s'_i \rangle$, each stored as separate data in the replay buffer. When a sufficient number of samples has been accumulated, a random batch of k transitions $\langle s_j, a_j, r_j, s'_j \rangle$, for $j = 1, \ldots, k$, is selected from this buffer for training.

To accommodate all possible reference states $x \in \mathcal{S}_i$, the algorithm prepares a $k' = k|\mathcal{S}_i|$ set of transition samples $\langle s_j, a_j, x, r_j, s'_j \rangle$. In cases where $|\mathcal{S}_i|$ exceeds 100, a random subset of 100 reference states x is chosen, thus limiting the number of samples to $k' = 100k$.

The following computations determine the predicted Q-values $Q^x_\theta(s_j, a_j)$ and the target Q-values. The latter are defined as

$$Q^x_{target}(s(t), a(t)) = r(s(t), a(t)) + (1 - a(t))\lambda(x) + \gamma \max_{v \in \mathcal{A}} Q^x_{\theta'}(s(t+1), v), \quad (4.10)$$

using a secondary network θ' for the $Q_{\theta'}$ values. This secondary network, initialized with the main network θ in weights, is periodically synced with the main network θ every 100 iterations.

The algorithm computes the loss function as the mean squared error:

$$L(\theta) = \frac{1}{k'} \sum_{j=1}^{k'} \left(Q^{(j)}_{\text{target}} - Q_\theta(s^{(j)}, a^{(j)}) \right)^2, \quad (4.11)$$

for each $j = 1, \ldots, k'$ sample. After calculating the loss function within the QWINN algorithm, the next step is to apply the Adam optimizer technique [49] for

Algorithm 2 QWINN Algorithm

```
**Input:** Discount parameter $\gamma \in (0,1)$, exploration parameter $\epsilon \in [0,1]$,
**Output:** Whittle index vector for all states
Initialize $s_0$ for all arms
for $n = 1 : n_{end}$ do
    Define action $a_i(n)$ through $\epsilon$-greedy policy for each arm $i$
    Get new states $s_i(n+1)$ and rewards $r_i(n)$ from states $s_i(n)$ and actions $a_i(n)$.

    Save each arm's data into separated memories
    if number of samples in memory $>$ threshold then
        for arm $i \in N$ do
            for $x \in S$ do
                Predict $Q$-values of sample batch $\langle s_i(n), a_i(n)\rangle$ using reference state $x$ for Whittle index with $Q_\theta^x$
                Compute target $Q$-value for sample $\langle s_j, a_j, x, r_j, s'_j\rangle$ as (4.10), using secondary network $Q_{\theta'}^x$
            end for
            Compute the mean square error loss function between $Q_\theta$ and $Q_{target}$ as Equation (4.11)
            Update the $\theta$ parameters of the $Q_\theta$ regressor through backpropagation using an Adam optimizer (Equations (3.3), (3.4), (3.5))
            Update the Whittle index for all states $x$ as (4.7)
            if $n\%50 = 0$ then
                Copy the $\theta$ parameters from the main $\theta$ neural network into the secondary $\theta'$ neural network
            end if
        end for
        Update Whittle index learning rate $\beta(n)$ as (4.9)
    end if
end for
```

backpropagation as introduced in Section 3.2.1. The Adam optimizer is a popular choice for training neural networks due to its adaptive learning rate mechanism, which adjusts the learning rate for each parameter based on the historical gradients.

After every 50 iterations, the algorithm updates the Whittle indices for all reference states $x \in \mathcal{S}_i$ following Equation (4.7) and learning rate $\beta(n)$ from (4.9). The QWINN algorithm is presented in Algorithm 2.

In the following section we will analyze the convergence of the QWINN neural network to its optimal parameters, providing a comprehensive understanding of the algorithm's convergence properties.

4.2.1 Proof of convergence of QWINN algorithm

In the following analysis, we investigate the local convergence properties of the DQN algorithm used in QWINN. We strengthen a necessary condition for local minima in optimization theory to an assumption and apply stochastic approximation theory to elucidate DQN's behavior in a neighborhood of a local minimum under specific conditions.

4.2.1.1 Preliminary Considerations

Consider the sequence $\tilde{\theta}_m = \theta_{T_n}$, defined for each $T_n \leq m < T_{n+1}$, where $T_n \uparrow \infty$. We specify that for some n, the parameter $\theta^*_n := \theta_{T_n}$ lies in a bounded neighborhood around a local minimum θ^* of the Bellman error function $\mathcal{E}(\cdot, \theta^*)$, defined as:

$$\mathcal{E}(\theta, \tau) := \mathbb{E}\left[\left\|Q^x_\theta(s,a) - (1-a)(r_0(s)+\lambda) - ar_1(s) - \gamma \max_{v \in A} Q^x_\tau(s', v)\right\|^2\right].$$

Let $\nabla_1 \mathcal{E}, \nabla^2_1 \mathcal{E}$ denote the gradient and the Hessian of $\mathcal{E}$ with respect to the first argument alone. Assuming that the Hessian $\nabla^2_1 \mathcal{E}(\theta^*, \theta^*)$ is positive definite, the inverse function theorem ensures a locally defined, bijective mapping $F(\tau) = (\nabla_1 \mathcal{E}(\cdot, \tau))^{-1}(\mathbf{0})$, where $\mathbf{0}$ is the zero vector, in a neighborhood of θ^*. Let us introduce an additional assumption.

Assumption 4.1. *$F(\tau)$ is locally a contraction around the equilibrium point (θ^*, θ^*).*

4.2.1.2 Local Convergence

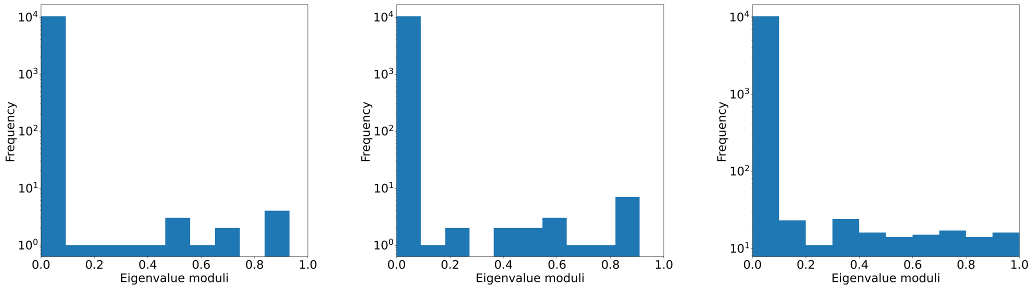
Theorem 4.2. *For a θ^* as above, under Assumption 4.1 and the condition that the Hessian $\nabla^2_1 \mathcal{E}(\theta^*, \theta^*)$ is positive definite, the DQN algorithm exhibits local convergence to an open ball of radius $\frac{2\epsilon}{1-\alpha}$ centered at θ^*, for $T_{n+1} - T_n$ sufficiently large.*

Proof. By Assumption 4.1, $F(\theta^*)$ is locally a contraction with some factor $0 < \alpha < 1$, and assuming that ϵ is within the specified bounds, consider $F_n(\theta^*_n) := (\nabla_1 \mathcal{E}(\cdot, \theta^*_n))^{-1}(\mathbf{0})$. For $T_{n+1} - T_n$ sufficiently large, $\|\theta^*_{n+1} - F_n(\theta^*_n)\| < \epsilon$. By continuity, for θ in the ϵ-neighbourhood of θ^*, $\|F_n(\theta^*_n) - F(\theta^*)\| < \epsilon'$, for some ϵ' that we take to equal ϵ without loss of generality. Then

$$\begin{aligned} \|\theta^*_{n+1} - \theta^*\| &\leq \epsilon + \|F_n(\theta^*_n) - F(\theta^*)\| \\ &\leq \epsilon + \|F_n(\theta^*_n) - F(\theta^*_n)\| + \|F(\theta^*_n) - F(\theta^*)\| \\ &\leq 2\epsilon + \alpha\|\theta^*_n - \theta^*\|. \end{aligned}$$

Iterating, it follows that θ^*_n approaches the δ-ball centred at θ^*. This establishes the local convergence of the DQN algorithm within a neighborhood B of θ^* to a δ-neighborhood thereof.

Figure 4.1 Histogram of eigenvalue moduli of $-(\nabla_1^2\mathcal{E}(\theta^*,\theta^*))^{-1}\nabla_2\nabla_1\mathcal{E}(\theta^*,\theta^*)$ for the circular (left), restart (middle) and deadline scheduling (right) problems.



4.2.1.3 Observations and Practical Implications

One of the key assumptions we have introduced in Theorem 4.2 is that $F(\tau)$, defined as $F(\tau) = (\nabla_1\mathcal{E}(\cdot,\tau))^{-1}(\mathbf{0})$, is locally a contraction around the point θ^*. This assumption is supported by numerical analysis, which shows that in our examples, $F(\tau)$ is indeed a contraction around θ^*. However, finding sufficient general conditions for this contraction to hold falls outside the scope of this thesis.

We have performed numerical calculations to illustrate that this assumption holds in the problems considered in the thesis. In order to do so, we have proceeded as follows. Linearizing $\nabla_1\mathcal{E}(\theta,\tau)$ in the proximity (θ^*,θ^*) we get

$$\nabla_1\mathcal{E}(\theta,\tau) \approx \nabla_1\mathcal{E}(\theta^*,\theta^*)+\nabla_1^2\mathcal{E}(\theta^*,\theta^*)(\theta-\theta^*)+\nabla_2\nabla_1\mathcal{E}(\theta^*,\theta^*)(\tau-\theta^*)+o(\theta-\theta^*,\tau-\theta^*).$$

Since, $\nabla_1\mathcal{E}(\theta^*,\theta^*) = 0$, we get

$$(\theta-\theta^*) \approx -(\nabla_1^2\mathcal{E}(\theta^*,\theta^*))^{-1}\nabla_2\nabla_1\mathcal{E}(\theta^*,\theta^*)(\tau-\theta^*).$$

To have a contraction (locally) around the equilibrium point, we need the spectrum of $-(\nabla_1^2\mathcal{E}(\theta^*,\theta^*))^{-1}\nabla_2\nabla_1\mathcal{E}(\theta^*,\theta^*)$ to be inside the unit disc in the complex plane around the origin.

The foregoing assumes that $\mathcal{E}$ is differentiable in the second argument, i.e., the target. More generally, a version of Danskin's theorem ensures existence of directional derivatives and the foregoing can be modified accordingly.

To calculate numerically the eigenvalues we have considered a random batch of training samples and compute the loss function. We then compute numerically the gradient of the loss function with respect to the first argument ($\nabla_1\mathcal{E}(\cdot,\cdot)$) and then again, the gradient thereof with respect to the first argument ($\nabla_1^2\mathcal{E}(\cdot,\cdot)$) and the second argument ($\nabla_2\nabla_1\mathcal{E}(\cdot,\cdot)$).

We have numerically verified that the condition on the spectrum for all the environments considered in Section 4.3.1 is satisfied. In Figure 4.1, we represent the eigenvalues obtained using the homogeneous environments in our work, for a neural network with size $((2,50),(50,100),(100,50),(50,2))$, that is, with 10402 parameters.

Furthermore, the practical application of this contraction depends on the intervals $T_{n+1} - T_n$ being sufficiently large. The larger these intervals are, the smaller the choice of δ can be. This observation may not be consistent with settings where $T_n = nT$ is used for some fixed $T > 0$ with decreasing step sizes. Nevertheless, it remains valid when a constant step size is employed, provided that T is sufficiently large.

4.3 Experimental Setup and Results for Discrete Models

In this section, we evaluate the performance of our algorithms, QWI and QWINN, against established frameworks in the field: traditional Q-learning, the Deep Q-Network (DQN), and NeurWIN, a neural network-based algorithm introduced by [59], designed to directly compute the Whittle indices' estimates $\lambda_\theta(s_n)$ of a problem using as an input for the neural network only the state whose index is to be computed.

The NeurWIN algorithm considers that a policy of indices that achieves optimal rewards for RMABP is equivalent to Whittle's index policy. To obtain this policy, they define an activation function using a sigmoid function:

$$\sigma_m(f_\theta(s[t]) - \lambda) = \frac{1}{1 + e^{-m(f_\theta(s[t]) - \lambda)}}, \tag{4.12}$$

where m is a sensitivity parameter. This function selects action $a = 1$ with probability $\sigma_m(f_\theta(s[t]) - \lambda)$ and $a = 0$ with probability $1 - \sigma_m(f_\theta(s[t]) - \lambda)$. For each mini-batch of episodes, they randomly choose two states s_0 and s_1, with s_0 serving as the fixed value $\lambda = f_\theta(s_0)$ and s_1 as the initial state. Multiple episodes are generated within each mini-batch, recording the actions and states visited based on the policy defined by Equation (4.12). The gradients h_e are calculated as follows:

$$h_e \leftarrow \begin{cases} h_e + \nabla_\theta \ln\left(\sigma_m(f_\theta(s[t]) - \lambda)\right) & \text{if } a[t] = 1 \\ h_e + \nabla_\theta \ln\left(1 - \sigma_m(f_\theta(s[t]) - \lambda)\right) & \text{if } a[t] = 0 \end{cases} \tag{4.13}$$

for each sample at time t within the episode. After all episodes in the mini-batch are completed, the neural network parameters are updated as follows:

$$\theta \leftarrow \theta + L_b \sum_e (G_e - \bar{G}_b) h_e, \tag{4.14}$$

where G_e represents the discounted net rewards, G_b is the average of these rewards, and L_b is the learning rate.

In order to compare the convergence speed of all these algorithms to the desired policy, we have decided to represent each NeurWIN episode update as a unique transition, so that at each iteration all algorithms are updated simultaneously. It is also important to highlight that, unlike in QWI and QWINN, the training and execution phases are separate in NeurWIN, i.e., we first need to learn the indices of each arm separately. This is showcased in the index Figures of Section 4.3.3.1, where the indices learned by NeurWIN for all settings are the same, unlike QWI and QWINN, which have different learning processes for each setting.

4.3.1 Description of Test Environments

We base our research in the context of three different Restless Multi-Armed Bandit (RMAB) problems: the “restart problem”, as proposed in [61]; the “deadline scheduling problem”, detailed in [62]; and the “circular environment problem”, explored in [58].

A significant aspect of these selected RMAB problems is their analytical tractability in terms of Whittle index computation. For each of these problems, we can calculate analytical expressions of the Whittle Index, allowing us to assess the accuracy of the Whittle index estimates produced by the algorithms under consideration.

In Section 2.4 we explored the computation of these indices given full knowledge of the problem’s dynamics.

4.3.1.1 Restart Problem

The restart problem presents an interesting scenario in the context of Restless Multi-Armed Bandit (RMAB) problems, characterized by a state space $\mathcal{S}_i = 0, 1, 2, 3, 4$. In this model, an active action ($a = 1$) always returns the arm to the initial state, denoted by $P^1(0, s) = 1$. In contrast, the transition probabilities under a passive action ($a = 0$) are represented by

$$P^0 = \begin{pmatrix} 1-x & x & 0 & 0 & 0 \\ 1-x & 0 & x & 0 & 0 \\ 1-x & 0 & 0 & x & 0 \\ 1-x & 0 & 0 & 0 & x \\ 1-x & 0 & 0 & 0 & x \end{pmatrix},$$

where the reward function is $R_0(s) = y^{s+1}$ for passive actions as opposed to $R_1(s) = 0$ for active actions. With parameters set to $x = y = 0.9$ and a discount factor of $\gamma = 0.9$, the Whittle index values computed for each state are $\lambda(0) = -0.9$, $\lambda(1) = -0.7371$, $\lambda(2) = -0.5373$, $\lambda(3) = -0.3188$, and $\lambda(4) = -0.0939$. These are indicative of a policy that favors keeping all arm states near 0 to maximize rewards.

An interesting feature of this problem framework is its ability to accommodate a variety of real-world settings. For example, in a manufacturing environment, machinery and equipment subject to wear and tear can be modeled using this RMAB framework. The state space reflects different levels of machine degradation, with proactive maintenance (active action) restoring optimal operating conditions, thereby optimizing productivity while balancing maintenance costs.

Similarly, in healthcare management for chronic conditions, states can represent the severity of a patient's health, with interventions ranging from intensive treatments (active actions) to outpatient care (passive actions). The goal is to manage healthcare resources to improve patient outcomes while managing the trade-offs between intervention costs and health benefits.

In the case of IT systems management, states could denote different levels of system performance or security, with active actions such as system reboots or upgrades ensuring optimal operational status. Passive actions, on the other hand, involve continuous monitoring with the potential risk of performance or security degradation. This scenario illustrates the need for strategic decision-making to maintain system integrity while minimizing disruption.

4.3.1.2 Circular Problem

The circular problem introduces a new dynamic within Restless Multi-Armed Bandit (RMAB) problems characterized by a compact state space $\mathcal{S}_i = \{0, 1, 2, 3\}$. This model simulates cyclical processes in which active actions lead to either staying in the same state with a probability of x or moving to the next state with a probability of $1 - x$. On the other hand, passive actions maintain the current state with a probability of x or regress to the previous state with a probability of $1 - x$, creating a circular transition dynamic that ensures continuity within the state space. The transition probabilities are formalized as:

$$P^1 = \begin{pmatrix} x & 1-x & 0 & 0 \\ 0 & x & 1-x & 0 \\ 0 & 0 & x & 1-x \\ 1-x & 0 & 0 & x \end{pmatrix}, P^0 = \begin{pmatrix} x & 0 & 0 & 1-x \\ 1-x & x & 0 & 0 \\ 0 & 1-x & x & 0 \\ 0 & 0 & 1-x & x \end{pmatrix}.$$

The reward structure in this problem does not depend on the action taken, but only on the state, with $R(0) = -1, R(1) = R(2) = 0, R(3) = 1$. Given a discount factor $\gamma = 0.9$, the computed Whittle index values for each state, defined as $\lambda(0) = -0.4390, \lambda(1) = 0.4390, \lambda(2) = 0.8652, \lambda(3) = -0.8652$, suggest a strategy that favors keeping arms in state 3 to maximize rewards through passive actions and using active actions to move arms from state 2 to state 3.

This problem is notable for its sparse rewards, presenting several states with zero rewards, thus requiring strategic navigation and far-sighted policies to optimize the rewards. The model is relevant in practical application for a variety of real-world scenarios that involve cyclical processes: One of such applications is ecological succession and conservation, where the states within the model can be interpreted as different stages of ecological development. Active actions, similar to conservation efforts, aim to promote biodiversity by encouraging the regenerative cycle inherent in natural ecosystems, but may also reset it to an earlier stage, reflecting the cyclical process of destruction and regrowth in natural environments. Similarly, the model relates to health and fitness programs, where states correspond to different levels of health or fitness. This scenario underscores the need to balance vigorous activity with adequate rest to prevent burnout, and thereby navigate the cyclical path to achieving and maintaining optimal well-being.

Cyclical economic models further showcase the relevance of the model by reflecting the tides of economic conditions through recession, recovery, growth, and peak periods. Here, active and passive policy interventions are critical in steering the economy through its cycles, highlighting the model's utility in economic planning and analysis.

4.3.1.3 Deadline Scheduling Problem

To study the deadline scheduling problem, we will base our model on the work of [62], which explores its dynamics. The state space for each arm i, with $|\mathcal{S}_i| = 130$, contains two variables: deadline $T \in [0, 12]$ and service time $B \in [0, 9]$, giving place to states $s = (T, B)$, while certain states $(0, B \neq 0)$ remain inaccessible due to the constraints of the Markov chain dynamics. These variables, T and B, represent the remaining time and the remaining workload to complete a task, respectively.

The model assumes that if an arm i has no assigned task, it is in a resting state $(0, 0)$. Conversely, the engagement of an arm with a task is characterized by its state $s_i(n) = (T_i(n), B_i(n))$, which is affected by an action $a_i(n)$ as:

$$s^i_{n+1} = \begin{cases} (T_i(n) - 1, (B_i(n) - a_i(n))^+) & \text{if } T_i(n) > 1, \\ (T, B) \text{ with prob. } P(T, B) & \text{if } T_i(n) \leq 1, \end{cases}$$

where the operator $b^+ = \max(b, 0)$ ensures the non-negativity of the workload variable.

In scenarios where $T = 1$, symbolizing the impending deadline, the system transitions to a new state, including the possible resting state of $(0, 0)$, chosen uniformly at random. The decay of B depends only on the activation of the arm, while T decays invariably. Failure to complete a task in time, represented by the state $(T = 1, B > 0)$, imposes an additional penalty, represented by the function $F(B_i(n) - a_i(n)) = 0.2(B_i(n) - a_i(n))^2$, in addition to the fixed activation cost $c = 0.8$. The reward function is thus expressed as:

$$R(s_i(n), a_i(n)) = \begin{cases} (1 - c)a_i(n) & \text{if } B_i(n) > 0, T_i(n) > 1, \\ (1 - c)a_i(n) - F(B_i(n) - a_i(n)) & \text{if } B_i(n) > 0, T_i(n) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

[62] extend their analysis by deriving an expression for the Whittle index, $\lambda(T, B, c)$:

$$\lambda(T, B, c) = \begin{cases} 0 & \text{if } B = 0, \\ 1 - c & \text{if } 1 \leq B \leq T - 1, \\ \gamma^{T-1} F(B - T + 1) - \gamma^{T-1} F(B - T) 1 - c & \text{if } T \leq B. \end{cases} \tag{4.15}$$

This model has multiple real life applications such as project management, where it assists in the efficient allocation of resources to ensure timely completion of tasks, thereby reducing the risk of project delays and budget overruns. Another key example is cloud computing, where it can streamline the scheduling of virtual machines or containers to improve computing resource utilization and avoid Service Level Agreement (SLA) violations. Similarly, in manufacturing, it enables efficient scheduling of production lines to ensure that products meet their production deadlines. Finally, in healthcare, the model provides a framework for optimizing appointment and resource scheduling to maximize healthcare services and minimize patient wait times, with penalties for delays emphasizing the critical role of providing on-time healthcare.

4.3.2 Evaluation Metrics and Benchmarks

In our numerical analysis of Restless Multi-Armed Bandit (RMAB) problems, it's crucial to recognize that the Whittle index heuristic, while a powerful tool, does not always yield the optimal policy. Instead, it is considered a suboptimal policy for these problems. On the other hand, this suboptimality gap in many cases can be considered negligible since Whittle's index heuristic is asymptotically optimal as the number of arms increases [63]. As we discussed in Section 2.2.2, MAB problems suffer from the curse of dimensionality, where the state space grows exponentially with the number of arms: a problem with one state space per arm with $|\mathcal{S}_i| = 4$ becomes $|\mathcal{S}| = 1048576$ states with only 10 arms. With 20 arms, this number increases to 10^{12}, equivalent to the number of galaxies in the observable universe [64]. A rigorous evaluation of the performance of an algorithm against the optimal policy computed by value iteration is therefore infeasible. Instead, we will use the theoretical Whittle index policy of the problems discussed in Section 4.3.1 as a baseline for the optimal policy.

The value function at any given state can be formulated as the expected sum of discounted rewards, following the Equation (2.6). Given these limitations, our approach to evaluating an algorithm's value function focuses on a stochastic evaluation of its policy.

During each evaluation epoch, we examine the policy $\pi(t)$ adopted by each algorithm under consideration, including QWI, QWINN, Q-learning, DQN, a random policy, and the theoretical Whittle index policy that serves as the baseline. Over $N_e = 10$ evaluation runs, we fix our analysis to $i \in [1, 500]$ random initial states s_i, which are applied consistently to all algorithms to ensure coherence. Over

$n_{iter} = 50$ iterations for each initial state $\mathbf{s}_i$, we aggregate the rewards $R(\mathbf{s}(n), \mathbf{a}(n))$ according to the policy $\pi(t)$, resulting in the total discounted reward

$$R_i = \sum_{n=0}^{n_{iter}} \gamma^n R(\mathbf{s}(n), \mathbf{a}(n)).$$

The next step is to average these discounted rewards over all initial states to derive $R = \frac{1}{500} \sum_{i=1}^{500} R_i$. This process is repeated for each evaluation $n_e \in N_e$, yielding a vector of summed discounted rewards $\mathbf{R} = \{R_e\}, e \in [1, \ldots, N_e]$. Consequently, at each evaluation epoch t, the averaged discounted summed reward $R(t) = \frac{1}{N_e} \sum_{e=1}^{N_e} R_e(t)$ is computed along with its confidence interval, denoted by the standard deviation $\sigma(\mathbf{R}(t))$.

Given the computational constraints, we consider this stochastic evaluation to be a reliable approximation of the mean-value function for all states s $\in \mathcal{S}$. Thus, given $V_\pi = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{s} \in \mathcal{S}} V_\pi(\mathbf{s})$, it is expected that

$$R(t) - \sigma(\mathbf{R}(t)) \leq V_{\pi(t)} \leq R(t) + \sigma(\mathbf{R}(t)).$$

For a sufficiently small $\sigma(\mathbf{R}(t))$ it follows that $V_{\pi(t)} \approx R(t)$.

Tables 4.1, 4.2, and 4.4 provide a summary of the performance of each algorithm in terms of the relative error of the value function $V_{\pi(t)}$ in the last iteration with respect to the theoretical Whittle index policy, defined as:

$$\text{Relative Error} = \frac{|V_{\pi(t)} - V_{\text{Whittle}}|}{|V_{\text{Whittle}}|}.$$

This analysis is conducted for each of the three RMAB problems discussed in Section 4.3.1 and all the configurations of arms and states.

In the case of algorithms that compute a Whittle index (QWI, QWINN, and NeurWIN), emphasis is also placed on illustrating the convergence of these algorithms' estimates $\lambda(s)$ to the theoretical Whittle index values $\lambda^*(s)$. This serves as an alternative benchmark to check how close each of these algorithms is to Whittle's index policy. Note that it is not the absolute value of these indices that defines the heuristic, but the order of the indices themselves. That is, for a set of indices where each index has a large error with respect to its corresponding theoretical value, it is possible to still obtain a good policy as long as these indices keep the same ordering as the theoretical ones. However, such a situation can easily lead to suboptimal policies when each of the arms has slightly different dynamics, in which case an accurate measurement of these indices is essential. For this reason, we will introduce two metrics to show the accuracy of the Whittle index estimates as well as their ordering: The first metric, the cumulative distribution function (CDF) of the absolute error, serves as a quantifier of the estimation accuracy for the Whittle indices. This distribution represents the proportion of indices (on the Y-axis) whose absolute error is less than a given amount (shown on the X-axis).

On the other hand, the Spearman correlation coefficient [65] ρ is a non-parametric measure used to assess the strength and direction of the association between two ranked variables, defined as:

$$\rho = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)},$$

where d_i is the difference between the ranks of the two variables for each observation i, and n is the number of observations. In the context of the Whittle indices, we use ρ to assess the ordering of the estimated indices with respect to the theoretical indices. A value close to 1 indicates a strong positive association, that is, the estimated indices closely match the theoretical indices ordering. On the other hand, a value close to -1 indicates a strong negative association, suggesting that the estimated indices are in reverse order with respect to the theoretical indices. A value close to 0 indicates no association, suggesting that the estimated indices are not in any particular order with respect to the theoretical indices.

4.3.3 Detailed Analysis of Experimental Results

In the following section, our exploration is structured around three main categories within the RMAB framework: the effect of varying the number of arms, the effect of varying the number of states per arm, and the dynamics introduced by heterogeneous arms. Each category reveals on different aspects of algorithm performance and convergence characteristics in RMAB problems.

4.3.3.1 Scalability in the Number of Arms

In this section we addres the exponential growth of the total state space $|\mathcal{S}|$ as a direct consequence of increasing the number of arms, a phenomenon commonly referred to as "the curse of dimensionality". Our analysis compares the performance of index algorithms (QWI, QWINN, and NeurWIN) against DQN and tabular Q-learning to illustrate the implications of this exponential growth. In particular, we aim to highlight the efficiency and speed of convergence of these algorithms in scenarios with varying numbers of arms. Through experiments set within the "circular", "restart", and "deadline scheduling" environments with configurations $(N = 5, M = 2)$, $(N = 10, M = 5)$, and $(N = 20, M = 8)$, we intend to demonstrate how index algorithms can mitigate the curse of dimensionality by decoupling the state space. In particular, the application of tabular Q-learning becomes impractical in larger settings due to the sheer volume of states to explore ($N = 10$ implies $4^{10} \approx 10^6$ states in the "circular" problem and $5^{10} \approx 10^7$ states in the "restart" problem), making it infeasible for comprehensive analysis in certain scenarios.

In the following discussion, due to the significant difference in state space sizes, we will separate the discussion of the "restart" and "circular" problems from the "deadline scheduling" problem.

Figure 4.2 Whittle indices per state for the circular problem with $N = 5, M = 2$ and $|\mathcal{S}_i| = 4$ (left) and the restart problem with $N = 5, M = 2$ and $|\mathcal{S}_i| = 5$ (right)

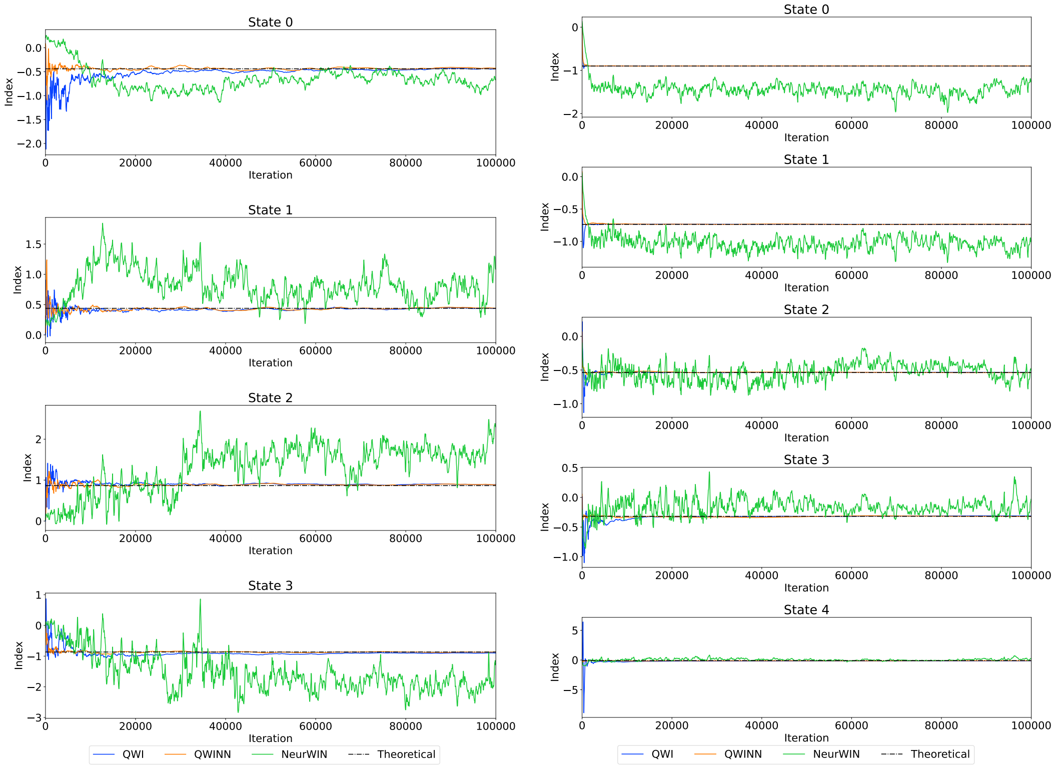
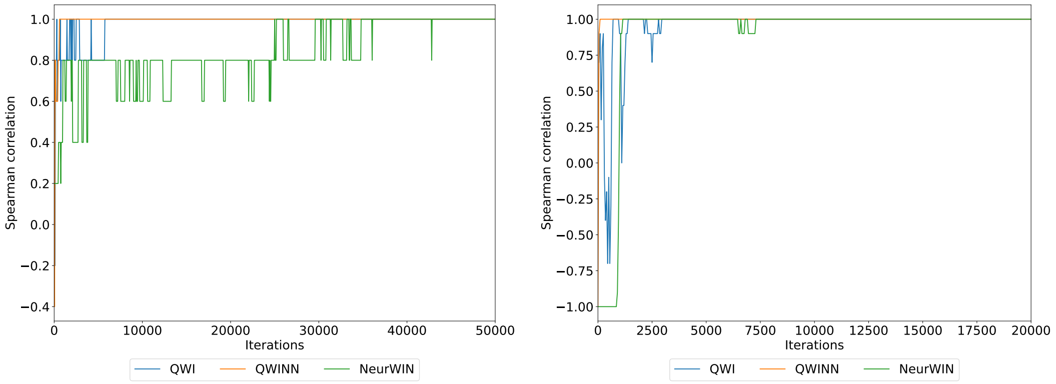


Figure 4.3 Spearman correlation coefficient for the circular problem (left) with $N = 5, M = 2$ and $|\mathcal{S}_i| = 4$ and restart problem (right) with $N = 5, M = 2$ and $|\mathcal{S}_i| = 5$



For both QWI and QWINN algorithm as well as NeurWIN, the Whittle indices computed by those algorithms do not depend on the number of arms or resources. As such, we will focus the analysis of those indices for the $(N = 5, M = 2)$ case, while the performance evaluation, specially for non-index algorithms, will be conducted for all three cases.

Circular and restart problems.

Figure 4.4 Average rewards for the circular problem with $N = 5, M = 2$ (top left), $N = 10, M = 5$ (top right) and $N = 20, M = 8$ (bottom) with $|\mathcal{S}_i| = 4$

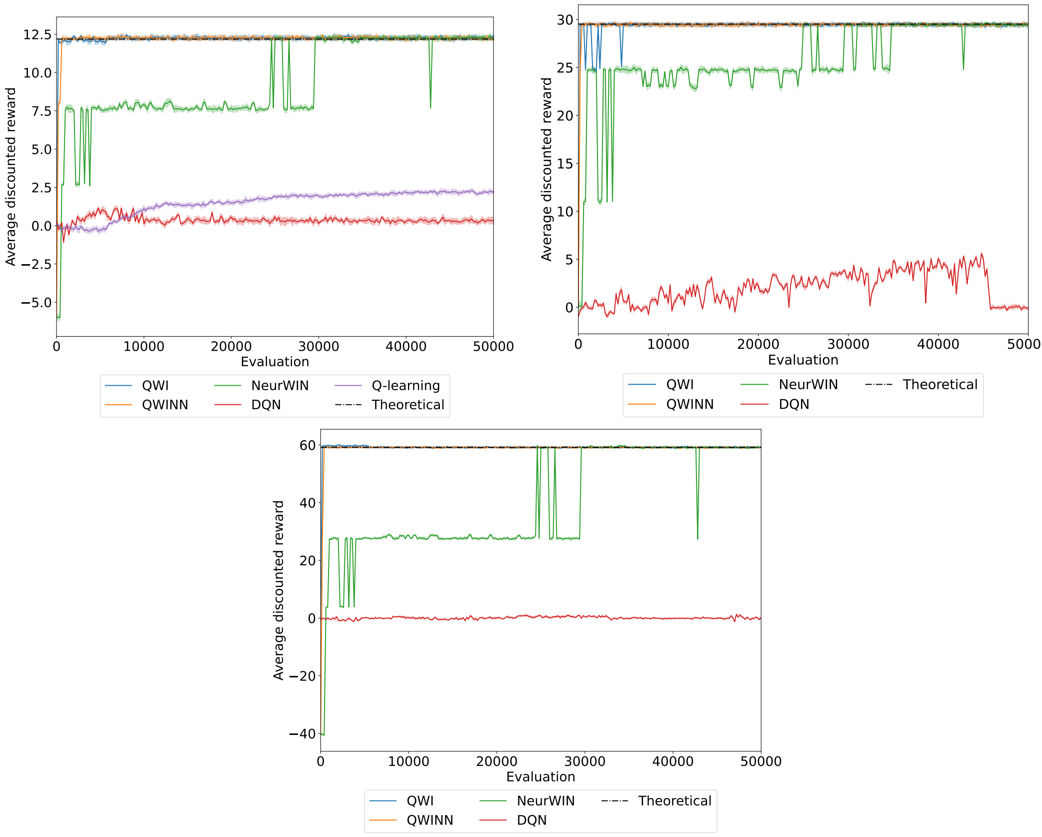


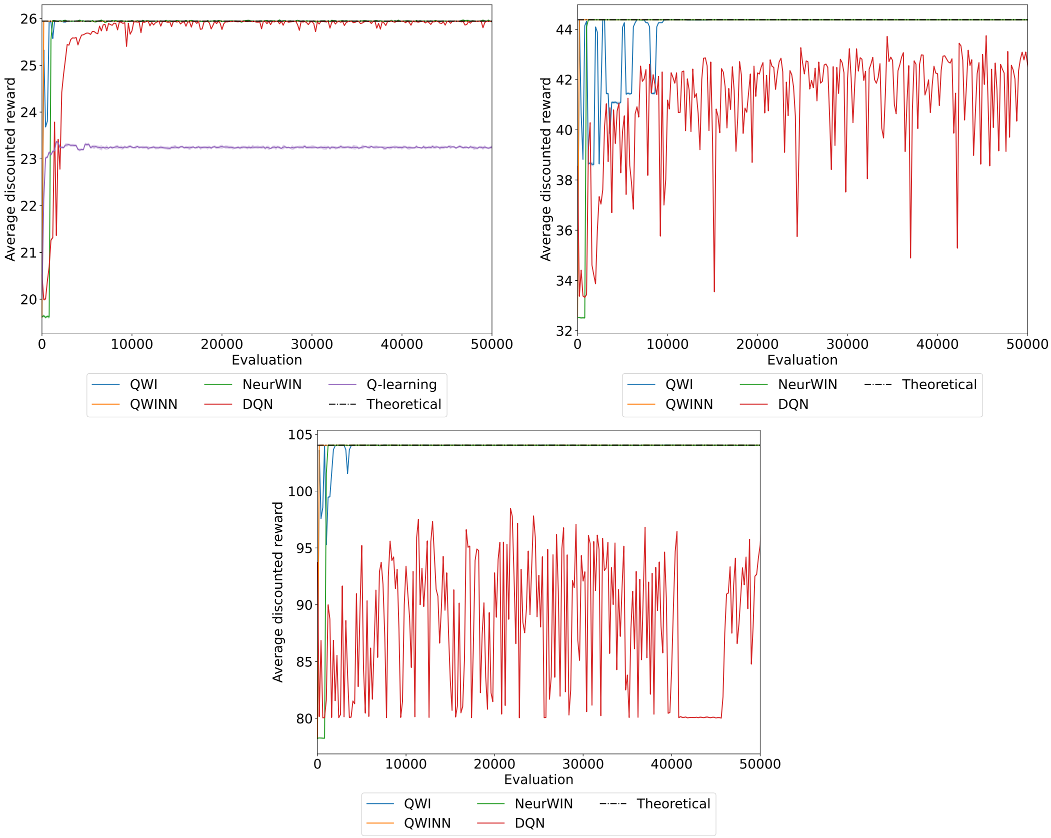
Figure 4.2 showcase the convergence plots of the Whittle indices for QWI, QWINN, and NeurWIN within the circular (left) and restart (right) problems, respectively.

A common observation across these environments is the quick convergence of both QWI and QWINN indices to their theoretical values, with QWINN outperforming QWI in convergence speed due to its ability to generalize information from other states. NeurWIN, on the other hand, shows a significant divergence from the theoretical values and lacks smooth convergence due to its optimization objective of only maximizing rewards through an indexing policy, supposedly consistent with Whittle's indexing policy. This often leads to instabilities in the computation of the Whittle index, which does not necessarily match the theoretical values.

Figure 4.3 illustrates the Spearman correlation for the circular and restart problems in QWI, QWINN, and NeurWIN. While all algorithms converge quickly to an optimal ordering for the restart problem (Figure 4.3 right), shown as $\rho = 1$, the circular problem (Figure 4.3 left) shows a noticeable misalignment of the indices for NeurWIN, which is only corrected after approximately 30000-35000 iterations.

The performance metrics, shown in Figures 4.4 and 4.5 for the circular and restart problems respectively, illustrate the performance of the algorithms. QWI

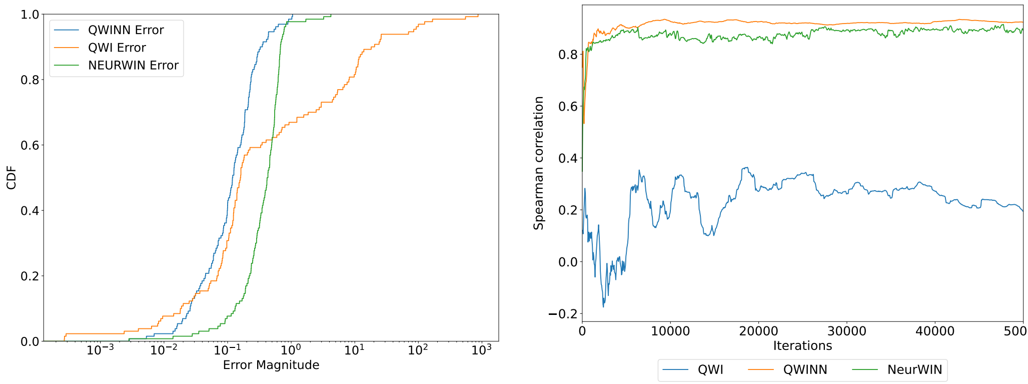
Figure 4.5 Average rewards for the restart problem with $N = 5, M = 2$ (top left), $N = 10, M = 5$ (top right) and $N = 20, M = 8$ (bottom) with $|\mathcal{S}_i| = 5$



and QWINN consistently reach the optimal policy within the first few hundred iterations. This is particularly noticeable in the circular problem (Figure 4.4), where, in contrast, NeurWIN is stuck with a suboptimal policy for the first 25000 iterations until it reaches the optimal policy. However, it does not remain completely stable, as shown by the negative spike in performance at iteration 43000. Furthermore, neither Q-learning nor DQN are able to obtain an optimal policy. Remarkably, Q-learning shows a gradual improvement over DQN in Figure 4.4 top left, while DQN fails to learn a good policy. However, as we increase the number of arms (Figures 4.4 top right and bottom), and consequently the number of states exponentially, the tabular Q-learning training becomes infeasible. Conversely, DQN's instabilities increase for larger arm configurations.

In the restart problem (Figure 4.5), NeurWIN is also stuck with a suboptimal policy for the first few thousand iterations until it rapidly converges to the optimal policy, but only after the QWI and QWINN algorithms. DQN is able to obtain the optimal policy for the $N = 5$ arm setting (Figure 4.5 top left), while tabular Q-learning is stuck with a low performance policy, due to its inability to visit most of the states in the problem. In the $N = 10$ (Figure 4.5 top right) and $N = 20$ (Figure 4.5 bottom) settings, however, DQN is unable to converge to a stable policy

Figure 4.6 Cumulative Distribution Function of the absolute error in the Whittle indices (left) and Spearman's Rank Correlation Coefficient (right) for the Deadline Scheduling problem with $N = 5$, $M = 2$ and $|\mathcal{S}_i| = 130$



and its performance oscillates well below the rest of the algorithms.

This disparity between index and non-index algorithms increases as we move to larger state-space problems, such as the deadline problem.

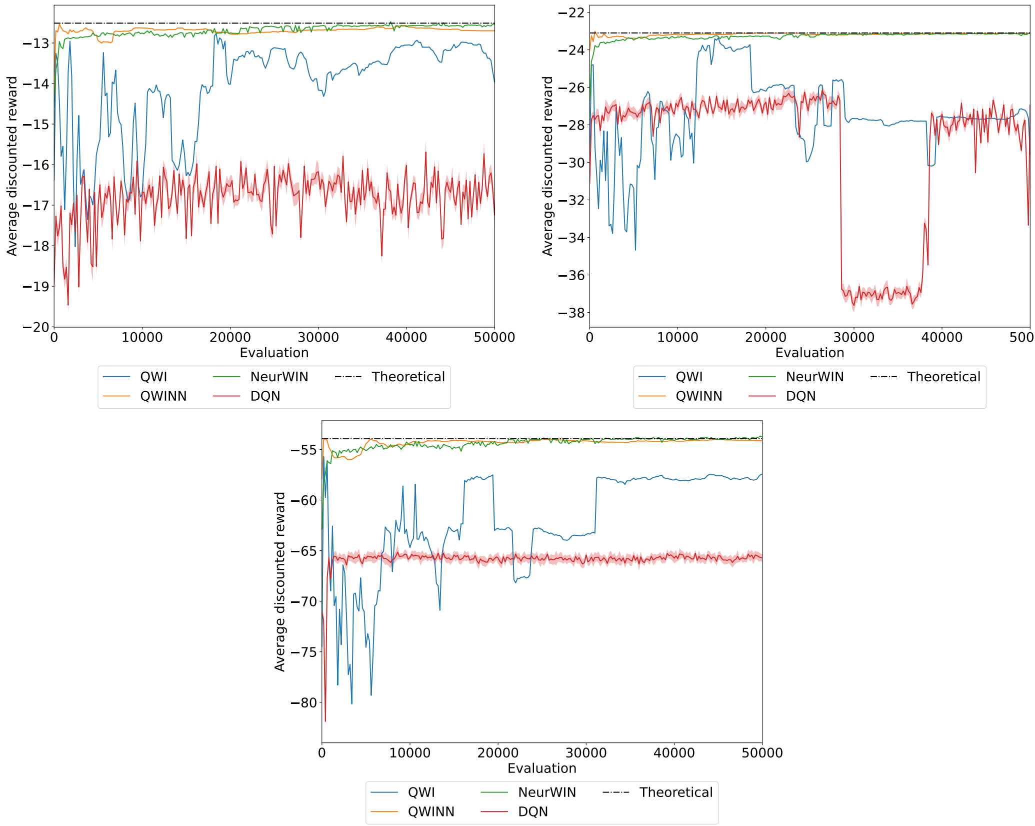
Deadline Scheduling Problem.

The Figure 4.6 left show the CDF of the absolute error in the Whittle indices for the deadline problem with $N = 5$, $M = 2$ and $|\mathcal{S}_i| = 130$ at the end of the training process. This figure shows the improved precision of QWINN, with most indices clustering around an error size of 10^{-1}. While QWI achieves similar accuracy for 60% of the indices, it also exhibits much larger errors for the remaining indices. In contrast, NeurWIN has most of its indices with an absolute error between 10^{-1} and 1. This difference highlights QWINN's ability to closely approximate a majority of states with remarkable precision, surpassing the sporadic performance of QWI and the broader error distribution of NeurWIN.

The evolution of the index ordering with respect to the theoretical Whittle index values, as shown in the Figure 4.6 right, show how none of the algorithms reach a correlation coefficient of $\rho = 1$, due to the harder difficulty of this larger environment. This phenomenon can also be attributed to the dense clustering of theoretical index values, where even small numerical inaccuracies can disrupt the precise ordering. In this case, both QWINN and NeurWIN exhibit a higher correlation coefficient than QWI, with QWINN showing a slight advantage over NeurWIN.

The performance metrics, which are derived from a stochastic evaluation similar to the one performed for the restart and circular problems, further illustrate the behavior of QWI, QWINN, NeurWIN, and DQN over different arm configurations. In the $N = 5$ setting (Figure 4.7 top left), both QWINN and NeurWIN show superior performance, closely mirroring the optimal policy, with NeurWIN showing a slight advantage in the second half of training by better ordering the most relevant states. Meanwhile, QWI, despite its challenges with the Spearman correlation coefficient,

Figure 4.7 Average rewards for the Deadline Scheduling problem with $N = 5, M = 2$ (top left), $N = 10, M = 5$ (top right) and $N = 20, M = 8$ (bottom) with $|\mathcal{S}_i| = 130$



achieves remarkable performance, surpassing the performance metrics provided by DQN.

In the $N = 10$ and $N = 20$ configurations (Figures 4.7 top right and bottom), QWINN and NeurWIN maintain parity in their performance, converging to the optimal policy threshold. DQN, on the other hand, has difficulties achieving a stable policy, with its performance falling at the $\sim 30,000$ iteration mark for the $N = 10$ setting. In the $N = 20$ setting, DQN's performance is stuck at a suboptimal policy, with no signs of improvement.

It is in these problems, where the increase of states is due to the increase of the number of MDPs, where index algorithms shine the most: classical methods are not able to adapt to problems with such a large state space, especially tabular Q-learning, which not only does not have enough samples to achieve convergence, but its computation becomes infeasible for cases with more MDPs. In such cases, QWINN is able to achieve the same optimal performance with similar or higher convergence speed as NeurWIN for all problems, while computing more accurate indices for all problems, while QWI, while being a tabular algorithm, is able to achieve surprisingly high performance, outperforming NeurWIN for problems with smaller $|\mathcal{S}_i|$.

Table 4.1 shows the relative error in the performance of the algorithms at the final iteration for the different problems according to the number of arms, with respect to the theoretical Whittle index policy. The results show that QWINN outperforms the rest of the algorithms in most cases, except for the circular problem with 20 arms, where QWI achieves a slightly lower relative error, and deadline scheduling problem, where NeurWIN shows better performance. NeurWIN also shows good performance in the restart environment, often matching QWINN and QWI's performance. DQN shows poor performance in all cases, with a relative error above 16% in all scenarios, and even exceeding 100% in some cases. Q-learning, where data is available, shows moderate performance, achieving a relative error below 10% only in the restart problem with 5 arms. Overall, QWINN and NeurWIN demonstrate the most robust and effective performance across the different environments and numbers of arms.

Table 4.1: Relative Error in Algorithm Performance at Final Iteration for Discrete Actions Problems according to Number of Arms in %

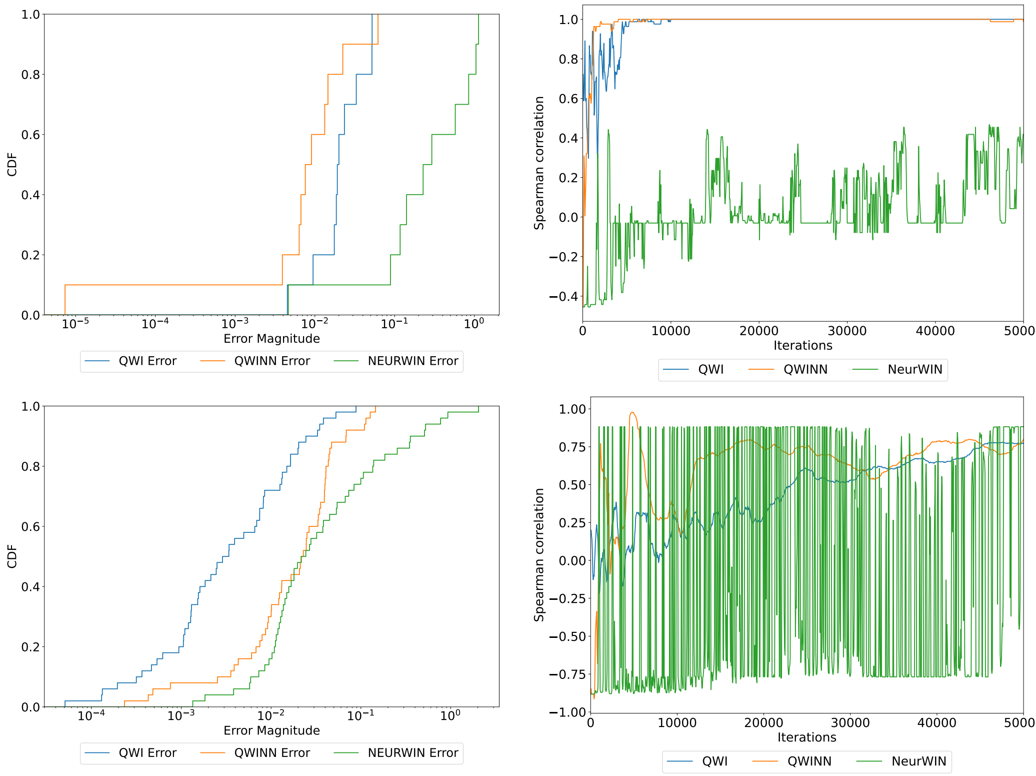
	QWI	QWINN	DQN	NeurWIN	Q-learning
Circular 5 arms	0.862 %	**0.120** %	97.080 %	0.574 %	81.923 %
Circular 10 arms	0.173 %	**0.100** %	80.606 %	0.586 %	
Circular 20 arms	**0.056** %	0.063 %	100.933 %	0.060 %	
Restart 5 arms	0.015 %	**0.011** %	0.177 %	**0.011** %	10.358 %
Restart 10 arms	**0.000** %	**0.000** %	2.685 %	**0.000** %	
Restart 20 arms	0.005 %	**0.003** %	23.071 %	0.004 %	
Deadline 5 arms	4.198 %	0.891 %	38.207 %	**0.223** %	
Deadline 10 arms	2.705 %	1.258 %	16.301 %	**0.052** %	
Deadline 20 arms	1.372 %	0.376 %	21.459 %	**0.229** %	

4.3.3.2 Scalability in the Number of States per Arm

The second part of our analysis focuses on the convergence efficiency of the algorithms as the number of states per arm is adjusted while keeping the number of arms fixed. This analysis is particularly insightful for evaluating the performance of neural network-based algorithms (QWINN, NeurWIN, and DQN) against QWI and tabular Q-learning in environments characterized by large state spaces. By studying environments such as “the circular environment” with $|\mathcal{S}_i| = \{4, 10, 50\}$, “the restart environment” with $|\mathcal{S}_i| = \{5, 20, 100\}$, and “the deadline scheduling problem” with $|\mathcal{S}_i| = \{130, 288\}$, using $\{(T = 12, B = 9), (T = 17, B = 15)\}$, in a $N = 5, M = 2$ setting, we aim to study the effect of the size of the state space per arm. Memory constraints require the exclusion of Q-learning from this analysis.

For the discussion of index convergence in larger state spaces, we will use the same metrics as in the “deadline scheduling” case in Section 4.3.3.1: the cumulative

Figure 4.8 Cumulative Distribution Function of the absolute error in the Whittle indices (left) and Spearman's Rank Correlation Coefficient (right) for the Circular problem with $N = 5$, $M = 2$ and $|\mathcal{S}_i| = 10$ (top) and $|\mathcal{S}_i| = 50$ (bottom)



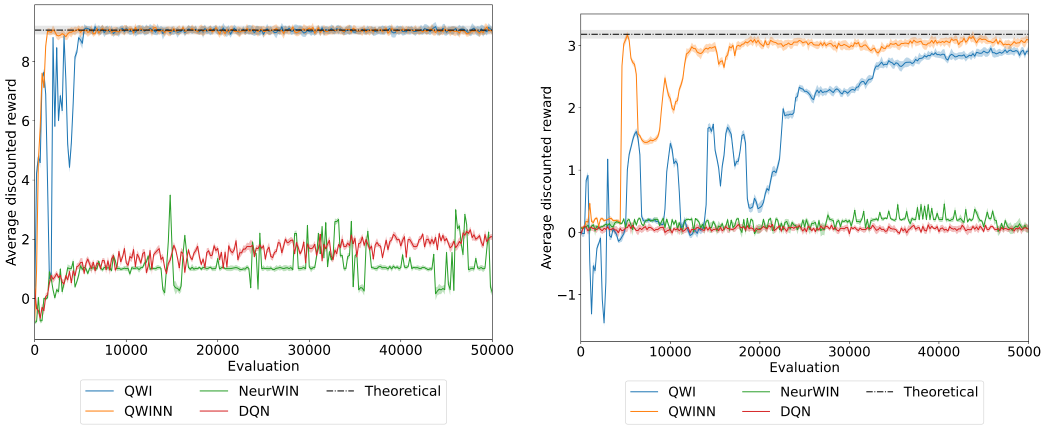
distribution function (CDF) of the absolute error of the Whittle index estimates and the Spearman correlation coefficient.

Circular problem.

In the circular problem (Figure 4.8), we observe the convergence of the Whittle indices for QWI, QWINN, and NeurWIN in scenarios with $|\mathcal{S}_i| = \{10, 50\}$. In the $|\mathcal{S}_i| = 10$ case, QWI and QWINN show remarkable precision in their index computation, with errors predominantly around 10^{-2} (Figure 4.8 top left). This accuracy extends to the ordering of the indices, where both QWI and QWINN perfectly match Whittle's theoretical index policy after only 10,000 iterations (Figure 4.8 top right). NeurWIN, on the other hand, shows a broader error spectrum, ranging between 10^{-1} and 1, a discrepancy that, although seemingly minor, leads to a persistent misordering of states.

This situation evolves as we increase the state space per arm to $|\mathcal{S}_i| = 50$. QWI shows remarkable robustness, maintaining high precision over its indices with errors distributed between 10^{-4} and 10^{-1}, as shown in Figure 4.8 bottom left. On the other hand, QWINN shows a slight increase in error, mostly in the range between 10^{-2} and 10^{-1}. Interestingly, NeurWIN mirrors QWINN's error distribution for about 60% of its indices, albeit with a subset having larger errors than those observed

Figure 4.9 Average rewards for the Circular problem with $N = 5, M = 2$ and $|\mathcal{S}_i| = 10$ (left) and $|\mathcal{S}_i| = 50$ (right)



in QWI and QWINN. Crucially, the evaluation of the index ordering in Figure 4.8 bottom right shows that NeurWIN's indices manifest an almost random policy due to an unstable ordering, in stark contrast to the gradual increase of the Spearman correlation coefficient for QWI and QWINN, which steadily approach to 1.

The performance comparisons for the $|\mathcal{S}_i| = 10$ scenario in Figure 4.9 left, highlight the critical role of accurate index ordering. QWI and QWINN, which benefit from their precise index alignment, show optimal performance beyond 10000 iterations. In contrast, NeurWIN's misaligned indices lead to suboptimal performance. It significantly underperforms DQN, underscoring a failure to effectively capture the environment's properties.

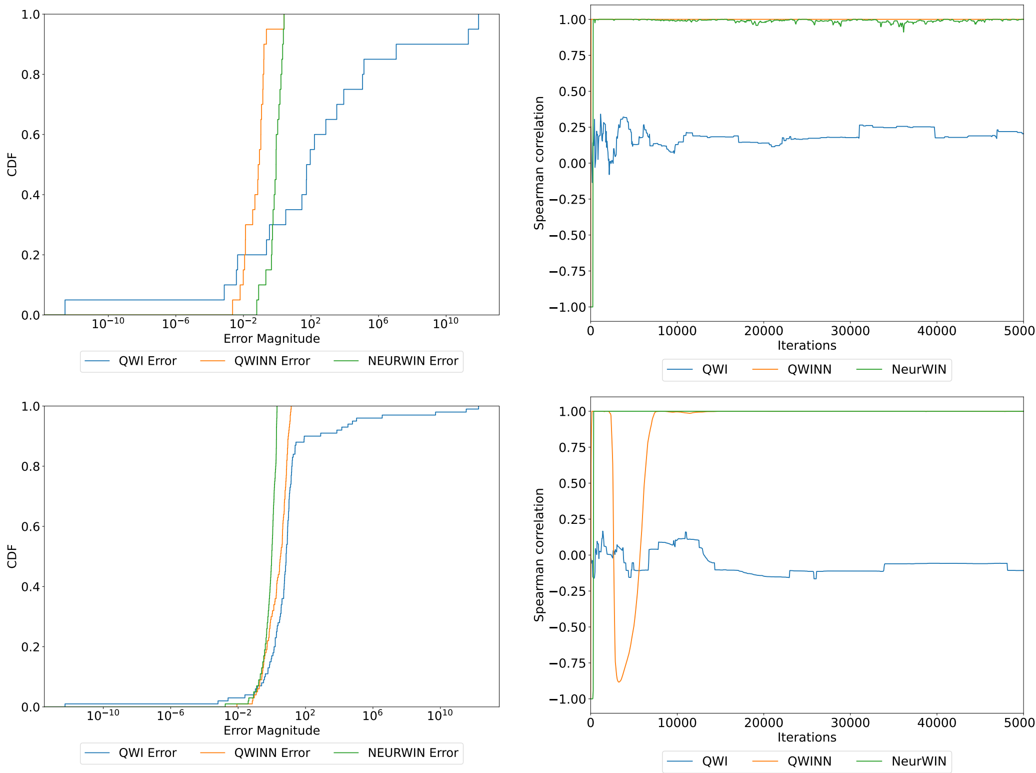
This divergence in optimality becomes even more pronounced in the $|\mathcal{S}_i| = 50$ setting (Figure 4.9 right). The initial index ordering advantage of QWINN over QWI, shown in Figure 4.8 bottom right, eventually narrows as QWI becomes more closely aligned with the optimal index ordering, leading to a significant performance increase, particularly evident in the first 20000 iterations, as shown in Figure 4.9 right. The performances of NeurWIN and DQN deteriorate to the level of a random policy, which is also true for DQN, which struggles against the huge size of the state space $|\mathcal{S}| = 50^5 = 312500000$. In contrast, QWI and especially QWINN show convergence to optimal policy performance.

Compared to the indices' ordering for $|\mathcal{S}_i| = 4$ in Figure 4.3 left and their respective performance in Figure 4.4 top left, it is clear that the initial instabilities of NeurWIN are amplified as we increase the number of states, while QWI and QWINN suffer smaller reduction in the speed of convergence. DQN, on the other hand, is not able to learn the properties of the problem correctly. It performs almost randomly throughout the training for larger configurations of $|\mathcal{S}_i|$.

Restart problem.

When studying the restart problem over different state space sizes per arm, the discussion of the convergence of the Whittle indices is reversed for QWI

Figure 4.10 Cumulative Distribution Function of the absolute error in the Whittle indices (left) and Spearman's Rank Correlation Coefficient (right) for the Restart problem with $N = 5$, $M = 2$ and $|\mathcal{S}_i| = 20$ (top) and $|\mathcal{S}_i| = 100$ (bottom)



and NeurWIN. For a state space of $|\mathcal{S}_i| = 20$ (Figures 4.10 top), QWI exhibits significantly larger errors, ranging from 10^{-2} to 10^{10}, in contrast to QWINN and NeurWIN, which largely maintain their index errors in the 10^{-2} range, as shown in Figure 4.10 top left. This result is expected given the increasing challenge of exploring all states within an arm as the state space expands, since a single active action transitions the current state to the first state in the environment, making it difficult to explore the later states. QWI's reliance on frequent state visits results in less accurate predictions for infrequently visited states, whereas the neural network frameworks within QWINN and NeurWIN mitigate this problem through their inherent ability to generalize across states.

This discrepancy in error translates directly into the ordering of the indices. More specifically, QWI shows a poor Spearman Rank Correlation Coefficient, fluctuating between 0 and 0.25 throughout the training, as shown in Figure 4.10 top right. Conversely, QWINN achieves a perfect ordering from the beginning, and NeurWIN also shows a high degree of ordering accuracy, with only occasional misplacements, indicating their robustness in preserving the theoretical index ordering.

A similar trend is observed in the cumulative distribution function (CDF) of

Figure 4.11 Spearman's Rank Correlation Coefficient for the first 5 states in the Restart problem with $N = 5$, $M = 2$ and $|\mathcal{S}_i| = 20$ (left) and $|\mathcal{S}_i| = 100$ (right)

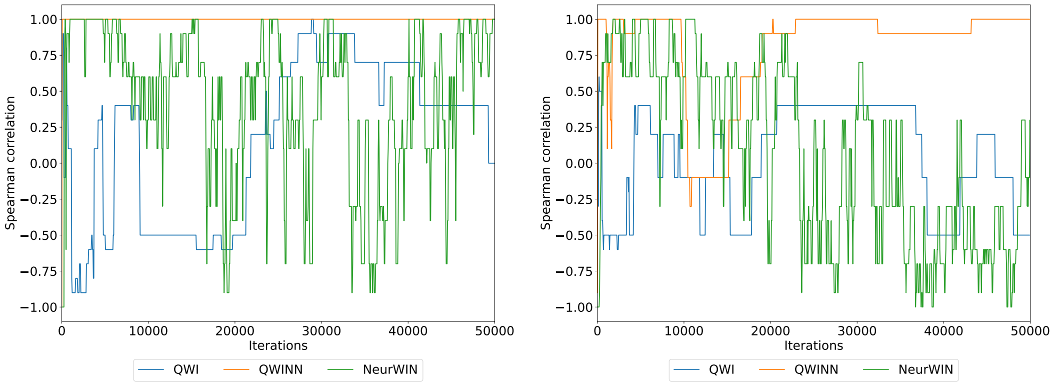
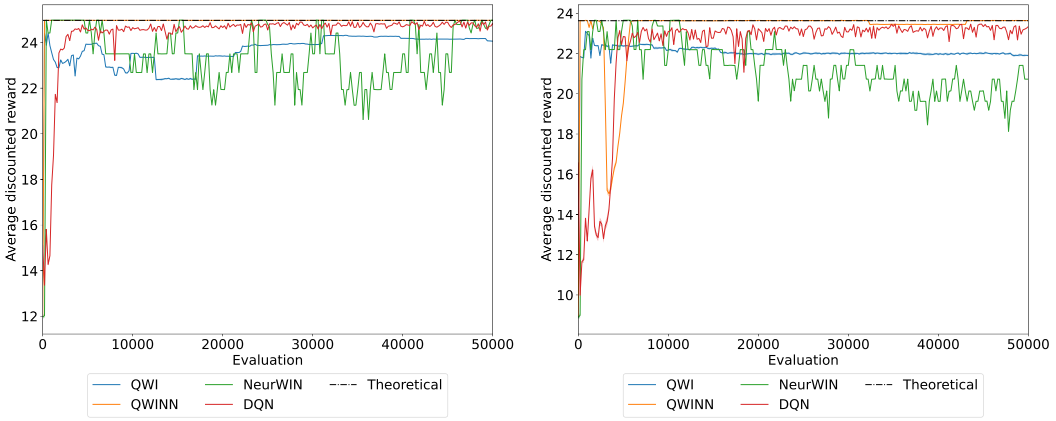


Figure 4.12 Average rewards for the Restart problem with $N = 5$, $M = 2$ and $|\mathcal{S}_i| = 20$ (left) and $|\mathcal{S}_i| = 100$ (right)



errors when expanding the state space per arm to $|\mathcal{S}_i| = 100$, as highlighted in Figure 4.10 bottom left. Both QWINN and NeurWIN, especially NeurWIN, show very small absolute errors, close to 10^{-2}. QWI, while showing comparable errors for the majority of its indices, sees about 15% of its indices fall to significant error magnitudes, escalating to 10^{10}. This error distribution affects the Spearman Rank Correlation Coefficient; in particular, NeurWIN's correlation quickly converges to 1, as shown in Figure 4.10 bottom right, with QWINN overcoming initial instabilities to achieve convergence around iteration 10000. QWI, however, reflects its previous performance problems with a large fraction of indices persistently out of order throughout training.

Despite this seemingly good ordering, a comparison of algorithmic performance on state spaces of $|\mathcal{S}_i| = 20$ and $|\mathcal{S}_i| = 100$, through Figures 4.12, reveals an erratic performance by NeurWIN, fluctuating between random and optimal rewards. In contrast, QWINN consistently converges to optimal performance, with DQN showing particular strength. Surprisingly, despite its index ordering challenges, QWI often outperforms NeurWIN. This phenomenon is due to the state transition dynamics of the problem, where actions transition the state directly to state 0, highlighting the importance of accurately ordering the first few states due to their frequent visits and central role in stochastic performance evaluation.

Further analysis of the Spearman coefficient, focusing only on the first 5 states in environments with $|\mathcal{S}_i| = 20$ and $|\mathcal{S}_i| = 100$ (Figures 4.11 left and right respectively), shows that while QWINN quickly reaches optimal ordering, both QWI and especially NeurWIN show fluctuating ordering, where both oscillate between a positive and a negative ρ, indicating alternating index orderings. In comparison, QWINN is still able to converge to a high ρ value, albeit more slowly in the $|\mathcal{S}_i| = 100$ setting.

To summarize, the escalation of state space complexity from the initial setting of $N = 5, M = 2, |\mathcal{S}_i| = 5$ widens the performance gap for QWI, which is limited by its inability to thoroughly explore and learn from the expanded state space. In addition, NeurWIN, despite its ability to maintain good index ordering for the majority of states, fails with the most critical states, leading to suboptimal performance that is even worse than that of QWI. QWINN, on the other hand, remains resilient to the increase in state space complexity, maintaining optimal index ordering and performance. DQN proves to be a surprisingly adaptable contender, able to handle the extended state space well and holding second place behind QWINN.

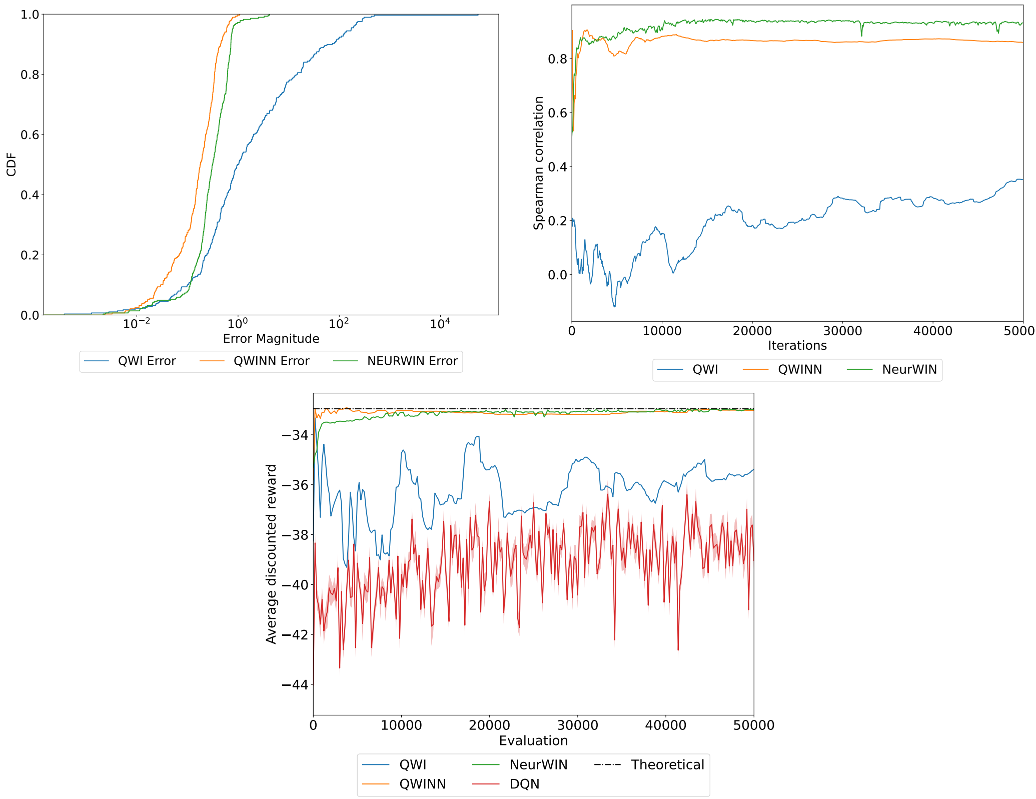
Deadline Scheduling Problem.

For the CDF error distribution of the deadline scheduling problem with a state space size of $|\mathcal{S}_i| = 288$ per arm, as shown in Figure 4.13 left, we see similar results to those in the estimates for $|\mathcal{S}_i| = 130$ (Figure 4.6 left). Both QWINN and NeurWIN manage to maintain absolute errors in the range of 10^{-2} to 1 for their index estimates, with QWINN showing slightly better accuracy than NeurWIN. By contrast, QWI exhibits significantly larger errors for about 85% of the states, with absolute errors ranging from 1 to 10^4. This continuation of the trend observed in Figure 4.6 left underscores the challenges QWI faces in comprehensively observing and learning from the full state space. Meanwhile, the neural network-based algorithms, QWINN and NeurWIN, effectively infer the characteristics of different states, demonstrating their superior generalization ability.

The comparison of Spearman's Rank Correlation for both state spaces, $|\mathcal{S}_i| = 130$ and $|\mathcal{S}_i| = 288$, illustrated in Figures 4.6 right and 4.13 right, shows how NeurWIN improves its index ordering accuracy with respect to QWINN in the $|\mathcal{S}_i| = 288$ setting, while both significantly outperform QWI. Interestingly, QWI shows a slight but clear improvement in index ordering over time in Figure 4.13 right, indicating a gradual alignment of its index ordering with theoretical expectations. Meanwhile, QWINN and NeurWIN reach their peak performance around iteration 10000, with the following iterations showing minimal fluctuations in their ρ values, indicating a stable index ordering.

The performance comparison, especially for the $|\mathcal{S}_i| = 288$ state space, shows a close result between QWINN and NeurWIN, as highlighted in Figure 4.13 bottom.

Figure 4.13 Cumulative Distribution Function of the absolute error in the Whittle indices (top left), Spearman's Rank Correlation Coefficient (top right) and Average rewards (bottom) for the Deadline Scheduling problem with $N = 5$, $M = 2$ and $|\mathcal{S}_i| = 288$



Initially, QWINN has a slight performance advantage over NeurWIN during the first 8000 iterations. NeurWIN's potentially better index ordering for certain states does not significantly affect overall performance when compared to other states that both algorithms prioritize accurately. QWI, despite its erratic performance, shows potential for long-term improvement. DQN, faced with the challenge of a large state space, struggles to generate an effective policy and lags behind, albeit with marginal signs of incremental improvement over time.

These results show the effect of increasing the number of states per arm $|\mathcal{S}_i|$ on the different algorithms: Similar to increasing the number of arms, algorithms that do not use index policies that decouple the state space, such as DQN, suffer a decrease in performance as a result of dealing with significantly larger problems. This problem is reduced in index algorithms such as QWI, QWINN and NeurWIN. However, as the state space in each arm increases, especially for problems with complicated exploration such as the restart problem, QWI suffers from performance degradation due to worse index estimation for most of these states. This is where the combined use of neural networks and index policy, as in QWINN and NeurWIN, achieves higher performance, as they can combine the reduction of problem complexity by decoupling the different MDPs, and at the same time obtain information about all states by inference of the state properties.

Table 4.2 shows the relative error in the performance of the algorithms at the final iteration for the different problems according to the number of states, with respect to the theoretical Whittle index policy. The results show that QWINN outperforms the rest of the algorithms in most cases, except for the circular problem with 10 states and deadline scheduling with 130 states, where QWI and NeurWIN respectively achieve a slightly lower relative error. NeurWIN shows good performance, often matching QWINN's performance, especially in the restart environment with 5 states and the deadline environments. DQN shows poor performance in most cases, except for the restart environment. Q-learning, where data is available, shows poor performance, and specially poor scalability in the state space, with an increased relative error when going from the 4 state circular problem to the 10 state case. Overall, QWINN demonstrate the most robust and effective performance across the different environments and numbers of states.

Table 4.2: Relative Error in Algorithm Performance at Final Iteration for Discrete Actions Problems according to Number of States in %

	QWI	QWINN	DQN	NeurWIN	Q-learning
Circular 4 states	0.862 %	**0.120** %	97.080 %	0.574 %	81.923 %
Circular 10 states	**0.601** %	1.064 %	74.427 %	85.013 %	101.726 %
Circular 50 states	8.653 %	**2.604** %	97.832 %	98.101 %	
Restart 5 states	0.015 %	**0.011** %	0.177 %	0.011 %	10.358 %
Restart 20 states	3.616 %	**0.008** %	1.000 %	0.046 %	
Restart 100 states	7.287 %	**0.037** %	1.574 %	12.277 %	
Deadline 130 states	4.198 %	0.891 %	38.207 %	**0.223** %	
Deadline 288 states	7.485 %	**0.205** %	14.110 %	0.217 %	

4.3.3.3 Performance in Heterogeneous Arms

The final part of our analysis deals with the complexity introduced by heterogeneous arms, where each arm has different dynamics. This aspect is crucial for understanding the implications of learning accurate Whittle indices within the indexing algorithms (QWI, QWINN, and NeurWIN). The main point of this section is to show that mere accuracy in ordering Whittle indices for individual arms may not be sufficient to achieve optimal global performance. For these heterogeneous environment problems, we consider the setting $N = 5, M = 2$, where each of the 5 MDPs has different dynamic parameters.

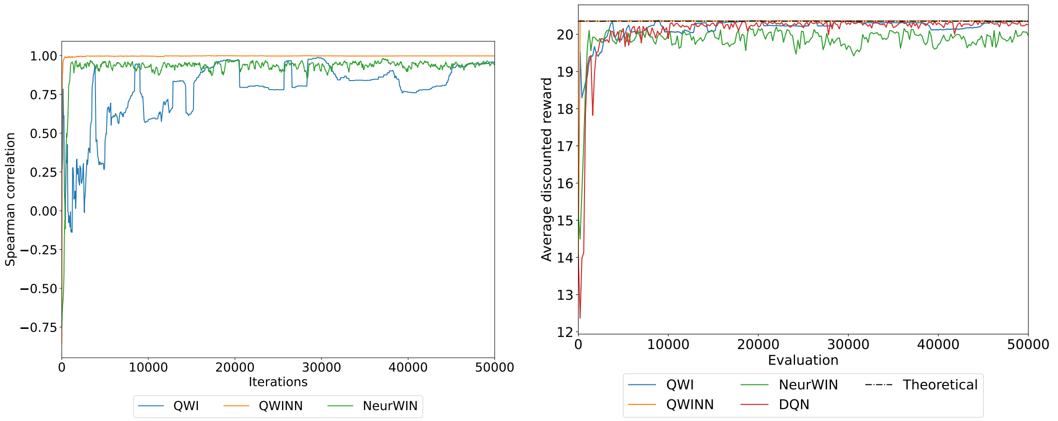
Restart problem.

In the case of the restart problem, we will focus on the $N = 5, M = 2, |\mathcal{S}_i| = 5$ setting, similar to the original case in Section 4.3.3.1 (Figures 4.3 right and 4.5 top left). In the original homogeneous setting, in addition to the impressive capability of QWI, QWINN, and NeurWIN to converge to the optimal Whittle index policy, it is worth

	State 0	State 1	State 2	State 3	State 4
$x = y = 0.9$	-0.9	-0.7371	-0.5373459	-0.31882516	-0.09391354
$x = y = 0.8$	-0.8	-0.5248	-0.2382848	0.02914796	0.26510922
$x = y = 0.7$	-0.7	-0.3577	-0.0597457	0.17455215	0.3499075
$x = y = 0.6$	-0.6	-0.2304	0.0333504	0.20520553	0.31272659
$x = y = 0.5$	-0.5	-0.1375	0.0690625	0.17803906	0.23380879

Table 4.3: Whittle index values for the restart problem with $N = 5, M = 2$ and $|\mathcal{S}_i| = 5$ and parameters $x = y = \{0.9, 0.8, 0.7, 0.6, 0.5\}$

Figure 4.14 Spearman's Rank Correlation Coefficient (left) and Average rewards (right) for the Restart problem with $N = 5, M = 2$ and $|\mathcal{S}_i| = 5$ and parameters $x = y = \{0.9, 0.8, 0.7, 0.6, 0.5\}$

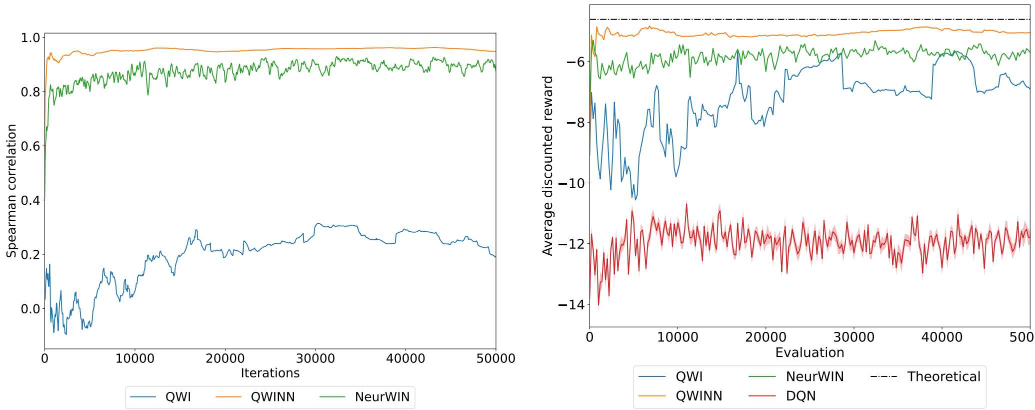


noting that DQN also showed remarkable convergence, achieving performance close to optimal. However, Q-learning lagged behind in comparison. Furthermore, in the homogeneous case, all three index algorithms (QWI, QWINN, and NeurWIN) were able to achieve a Spearman correlation coefficient of $\rho = 1$, indicating a perfect ordering of the Whittle index with respect to the theoretical values.

For the heterogeneous case of the restart environment, we consider a situation where each of the 5 MDPs has different transition and reward parameters. As described in Section 4.3.1.1, the restart problem is parametrized by the terms x, y that determine the passive transition probabilities and reward functions respectively, and in the previous sections they were set to $x = y = 0.9$. In this heterogeneous case, this values will be $x = y = \{0.9, 0.8, 0.7, 0.6, 0.5\}$, each for a different MDP. Varying these parameters results in the index values described in Table 4.3.

The evaluation of the Spearman correlation rank coefficient in the heterogeneous setting (Figure 4.14 left) illustrates the differences between the algorithms. NeurWIN, which previously achieved perfect correlation, now shows a slight decline in optimality, with a Spearman ρ close to 0.9. QWI faces significant instabilities due to the narrow margin separating the index values of subsequent states, especially the final ones of the environment, which requires a very high numerical

Figure 4.15 Spearman's Rank Correlation Coefficient (left) and Average rewards (right) for the Deadline Scheduling problem with $N = 5, M = 2$ and $|\mathcal{S}_i| = 130$ and processing cost $c = \{0.9, 0.7, 0.5, 0.3, 0.1\}$



precision for an accurate ranking. Remarkably, QWINN stands out as the only algorithm that maintains stable convergence to a ρ value of 1, reproducing the theoretical ordering of the Whittle indices.

In Figure 4.14 right, we compare the performance of QWI, QWINN, NeurWIN and DQN. The disparity between NeurWIN's performance in the homogeneous and heterogeneous scenarios becomes clear, with a clear deviation from the optimal rewards of the theoretical Whittle index policy. Despite QWI's lower overall ρ value, its performance surpasses that of NeurWIN, mainly due to the ranking problems affecting the less frequently visited states. QWINN, on the other hand, consistently reproduces the optimal rewards, demonstrating the algorithm's robust adaptability to heterogeneous environments. DQN impressively maintains a near-optimal policy, a continuity from its homogeneous performance, while Q-learning lags with sub-random policy effectiveness.

Deadline Scheduling Problem.

For the heterogeneous scheduling problem, we focus on a variety of processing cost values across multiple arms. The Whittle indices for the deadline scheduling problem, as defined in Equation (4.15), are closely related to the state variables - deadline time T, remaining workload B - and, in particular, the processing cost c associated with action execution. In this case, we assign different processing costs $c = \{0.9, 0.7, 0.5, 0.3, 0.1\}$ to each of the five MDPs, introducing a range of different Whittle indices within the $N = 5, M = 2, |\mathcal{S}_i| = 130$ setting for each MDP.

In the homogeneous case, previously discussed in Section 4.3.3.1, both QWINN and NeurWIN showed a high Spearman correlation coefficient. This translates into a high performance, with NeurWIN almost reaching the optimal policy, as shown in Figure 4.7 top left. However, moving to the heterogeneous environment increases the importance of precision in index computation. Since each MDP has a

Table 4.4: Relative Error in Algorithm Performance at Final Iteration for Discrete Actions Problems (homogeneous vs. heterogeneous) in %

	QWI	QWINN	DQN	NeurWIN	Q-learning
Restart Homogeneous	0.015 %	**0.011** %	0.177 %	**0.011** %	10.358 %
Restart Heterogeneous	0.044 %	**0.043** %	0.403 %	1.441 %	20.735 %
Deadline Homogeneous	4.198 %	0.891 %	38.207 %	**0.223** %	
Deadline Heterogeneous	48.382 %	**9.495** %	155.679 %	22.054 %	

unique set of indices due to different processing costs, the algorithms are tasked with accurately capturing the subtle differences associated with this setting. Figure 4.15 left shows the Spearman's Rank correlation of QWI, QWINN and NeurWIN, demonstrating QWINN's superior capability to reproduce the theoretical Whittle indices ordering throughout training. NeurWIN falls short of QWINN's accuracy to the theoretical benchmarks, with QWI trailing even further in the Spearman's correlation coefficient metric.

The performance evaluation shown in Figure 4.15 right shows how QWINN's proficiency in index estimation on a global scale results in a policy that outperforms its counterparts, albeit without quite reaching optimal policy performance. NeurWIN manifests a noticeable optimality gap with respect to QWINN, reflecting the differences in index ordering. Interestingly, despite its suboptimal index ordering, QWI still achieves commendable performance, occasionally paralleling NeurWIN's performance. This suggests that, as with the heterogeneous restart problem previously discussed, most of QWI's index ordering discrepancies are due to states that are rarely encountered during both the training and evaluation phases. In contrast, DQN struggles to surpass the baseline of random policy effectiveness.

Table 4.4 shows the relative error in the performance of the algorithms at the final iteration for the different problems in homogeneous and heterogeneous environments, with respect to the theoretical Whittle index policy. The results indicate that QWINN is the most robust algorithm for heterogeneous environments. While all algorithms experience an increase in absolute error, particularly in the deadline scheduling heterogeneous case, QWINN's increase is significantly less pronounced. For example, NeurWIN, which had the best performance in the homogeneous deadline case with a relative error of 0.22%, now has an error of 22.054% in the heterogeneous case. In contrast, QWINN's error only increases from 0.891% to 9.495%. This robustness is also seen in the restart heterogeneous environment, where QWINN maintains the lowest error at 0.043%. DQN and Q-learning show substantial increases in error across heterogeneous environments, with DQN's error reaching as high as 155.679% in the deadline heterogeneous case. Overall, QWINN demonstrates superior robustness and effectiveness in managing the increased complexity of heterogeneous environments compared to other algorithms.

Chapter 5

Continuous Action Models in Weakly Coupled MDPs

In the previous chapters, we introduced the Restless Multi-Armed Bandit Problem (RMAB), a complex framework in which multiple Markov Decision Processes (MDPs) with binary actions are deeply coupled. To this end, we introduced the novel algorithms QWI (Section 4.1) and QWINN (Section 4.2), which represent the state-of-the-art for both tabular and neural network approaches to computing Whittle indices, a key heuristic for decoupling the arms of the RMAB problem. This significant advance was shown in comparison to NeurWIN, the previous benchmark, alongside traditional stalwarts such as DQN and Q-learning. In particular, QWINN showed a better performance in terms of fast and accurate computation of Whittle indices across a range of problems varying in both the size of their state spaces and the number of arms, including scenarios characterized by arm heterogeneity.

However, the domain of binary actions, while complex, represents only a fraction of the variety of challenges encountered in real-world applications. Thus, our analysis now extends to the domain of continuous action problems. Applications as diverse as resource allocation in cloud computing, dynamic pricing, and renewable energy management require a transition from binary to continuous multi-action.

This path reveals a number of challenges that are unique to continuous action models. The inherent complexity of continuous actions introduces a new dimension of constraints-constraints that are no longer binary or categorical, but often nonlinear, leading to intricate relationships between the magnitude of actions, resource utilization, and their resulting effects.

5.1 Computation of Whittle Indices for Continuous Actions

Let us consider the Weakly Coupled MDP described in Section 2.2.4, characterized by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathbf{P}, \mathbf{R}, \mathbf{C}, N, B \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathbf{P}$ is the transition probability matrix, $\mathbf{R}$ is the reward function, $\mathbf{C}$ is the cost function, N is the number of arms, and B is the budget or amount of resources available at each time step.

In Section 2.4, we introduced the algorithms presented in [45] and [46] to compute the Whittle indices for discrete actions in the binary and multi-action cases respectively. In this section, we will extend these methods to the continuous action case, providing a numerical method to compute the Whittle indices for an arbitrary discretization in the action space.

To begin the computation of these indices, let us define the policy matrix Ω_0, a $n_{\text{states}} \times n_{\text{actions}}$ matrix, where each row indicates the action intended for a state, with all initial actions set to 0.

$$\Omega_0 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ \cdots & \cdots & \cdots \\ 1 & 0 & 0 \end{pmatrix}$$

Through iterative processes, we examine states not yet associated with the highest action, incrementing their actions to compute potential indices. This computation uses the transition probability matrix P_Ω, along with the reward and cost vectors R_Ω and C_Ω, based on the existing policy Ω_t. Given a target state, we adjust P'_Ω, R'_Ω, and C'_Ω to change the action for that target state according to the next policy Ω_{t+1}. We then compute the difference $\Delta_\Omega = P'_\Omega - P_\Omega$ and evaluate it for the specific target state: $\Delta_\Omega(s_{\text{target}})$, which corresponds to the row of s_{target} in the Δ_Ω matrix.

Following Equations (2.30) and (2.31), we define the value and cost functions as:

$$V = \left(\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} - \gamma P_\Omega \right)^{-1} \times R_\Omega \tag{5.1}$$

$$C = \left(\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} - \gamma P_\Omega \right)^{-1} \times C_\Omega \tag{5.2}$$

This process results in the index formula:

$$\lambda = \frac{f'_\Omega - f_\Omega}{g'_\Omega - g_\Omega} = \frac{R'_\Omega - R_\Omega + \gamma \Delta(s_{target}) \times V}{C'_\Omega - C_\Omega + \gamma \Delta(s_{target}) \times C}. \tag{5.3}$$

Once all indices are computed, the highest index value dictates the subsequent action, iteratively updating the policy matrix until it reaches the highest action for all states, denoted Ω_{end}.

Algorithm 3 Generalized Multi-action Adaptive-greedy Whittle Index Algorithm

```
Initialize policy matrix $\Omega_0$
while not all states have reached their highest action do
    for each state $s \in \{1, 2, \ldots, n_{\text{states}}\}$ do
        Update matrices $P'_\Omega$, $R'_\Omega$, $C'_\Omega$ based on new policy $\Omega_{t+1} := a'(s_{\text{target}}) = a + \delta$

        $\Delta_\Omega \leftarrow P'_\Omega - P_\Omega$
        $\Delta_\Omega(s_{\text{target}}) \leftarrow$ row of $s_{\text{target}}$ in $\Delta_\Omega$
        Define $V$, $C$ and $\lambda$ using Equations (5.1), (5.2), and (5.3) respectively.
        Update policy matrix $\Omega_t$ based on highest index value
    end for
    Update policy $\Omega_t \leftarrow \Omega_{t+1}$ for $s_{\text{target}}$ with highest index
end while
Output: List of indices $\lambda(s, a)$
```

$$\Omega_{\text{end}} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ \cdots & \cdots & \cdots \\ 0 & 0 & 1 \end{pmatrix}$$

Algorithm 3 extends the methods of Niño-Mora and Weber to a broader range of action discretizations, δ. Given $\delta = 1$, it falls back to Weber's Algorithm [46], while using $a_m = 1$ we obtain the original Niño-Mora's Algorithm [45].

5.1.1 Policy Heuristic for Continuous Actions

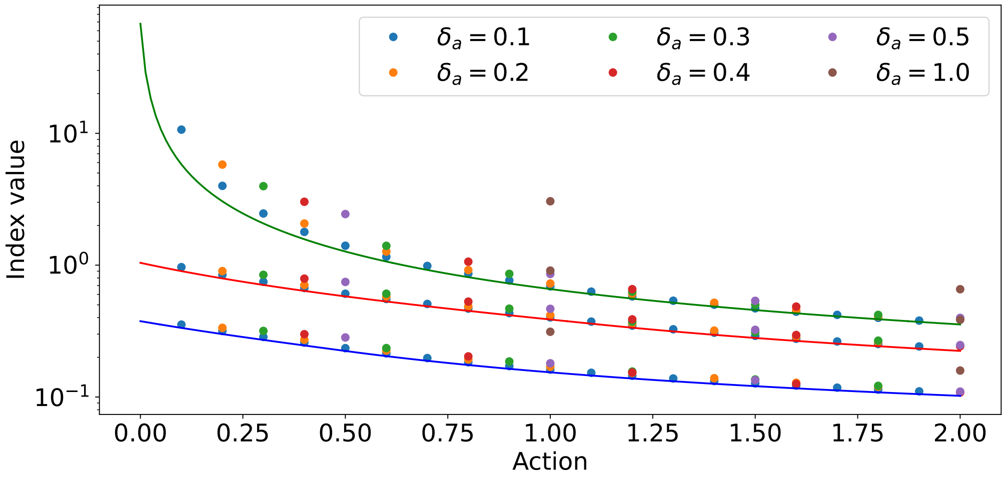
The heuristic for determining actions using the Whittle index in the context of continuous actions requires a more complex approach beyond the simpler approach inherent to the binary action scenario. In the traditional binary case, the methodology involves selecting the M projects with the highest indices. However, the continuous action framework requires an examination of the costs associated with each activation level in order to identify the optimal actions.

Consider Figure 5.1, which show the Whittle index computed using Algorithm 3 for different levels of discretization for a given problem. Given an index value λ, the action values of each curve represents the appropriate activation level.

In the relaxed version of the problem, resource consumption is averaged over time, allowing for scenarios where the total cost of the chosen action vector temporarily exceeds or falls below the resource bound B. In contrast, the original problem formulation enforces the constraint that the action vector $\mathbf{a}$ incurs a total cost $c(\mathbf{s}, \mathbf{a}) = B$.

Algorithm 4 introduces a heuristic for computing the actions associated with Whittle indices given a resource constraint B. This heuristic uses a binary search

Figure 5.1 Comparison of the Whittle Index for the Admission Control problem using different levels of discretization.



Algorithm 4 Continuous Action Whittle Index Heuristic

```
**Procedure:** GreedyIndexContinuousActions($\mathbf{s}, B, a_{\max}$)
Initialize minimum and maximum index values $\lambda_{\min}$ and $\lambda_{\max}$
**while** $|\lambda_{\max} - \lambda_{\min}| > \delta$ **do**
    $\lambda_{target} = \frac{\lambda_{\min}+\lambda_{\max}}{2}$
    **for** each MDP $i$ in the coupled problem **do**
        $\lambda(s_i, a_i), \quad \forall a_i \in \mathcal{A}_i$
        $a_i^* \leftarrow \arg\min_{a_i} |\lambda_{target} - \lambda(s_i, a_i)|$
    **end for**
    $\mathbf{a}^* = \{a_i^*\}$
    $C(\mathbf{s}, \mathbf{a}^*) = \sum_{i=1}^{N} C(s_i, a_i^*)$
    **if** $C(\mathbf{s}, \mathbf{a}^*) > B$ **then**
        $\lambda_{\min} \leftarrow \lambda_{target}$
    **else**
        $\lambda_{\max} \leftarrow \lambda_{target}$
    **end if**
**end while**
**Output:** $\mathbf{a}^*$
```

over the index space for a given state vector s, seeking to adapt the policy so that $c(\mathbf{s}, \mathbf{a}_{\mathbf{s}}(\lambda)) = B$.

Starting with boundary values $\lambda_{\min}$ and $\lambda_{\max}$, we estimate an initial target index $\lambda_{target} = \frac{\lambda_{\min}+\lambda_{\max}}{2}$. This serves as a preliminary estimate of the optimal index under the resource constraint for a vector of states s. For each individual MDP within the coupled problem, we evaluate all possible actions $a_i \in [0, a_{max}]$ to determine the index $\lambda(s_i, a_i)$, and then identify the action a_i^* that minimizes the absolute deviation from λ_{target}. Hence: $a_i^* = \arg\min_{a_i} |\lambda_{target} - \lambda(s_i, a_i)|$.

Once we have a vector of actions $\mathbf{a}^*$, if the total cost $c(\mathbf{s}, \mathbf{a}^*)$ exceeds B, it implies that the action selection is too big. To mitigate this, we adjust by increasing λ_{target} for the subsequent iterations, setting $\lambda_{\min} = \lambda_{target}$. Conversely, if the total cost is below B, indicating overly conservative actions, we increase the potential resource utilization by decreasing λ_{target}, setting $\lambda_{\max} = \lambda_{target}$.

This iterative process continues until the gap between $\lambda_{\max}$ and $\lambda_{\min}$ narrows significantly, at which point convergence to the optimal action given the resource constraints is achieved.

5.2 LPCA: Tackling Weakly Coupled MDPs with Continuous Actions

In the following sections, we will reintroduce the problem formulation for the Weakly Coupled MDP and present the algorithm for the Lagrange Policy for Continuous Actions (LPCA) [30], a reinforcement learning algorithm specifically designed for weakly coupled MDP problems with continuous action spaces. LPCA integrates a neural network-based framework (as introduced in Section 3.2.1) to study weakly coupled MDP using the Lagrange relaxation introduced in [43] to decouple the projects of the MDP, being able to study their dynamics independently of one another and effectively balancing resource constraints and individual project decisions. In Section 5.2.1, we will present the problem formulation for the LPCA algorithm, followed by a detailed explanation of the LPCA algorithm in Section 5.2.3. Finally, in Section 5.3.3, we will present the experimental results of the LPCA algorithm in a variety of scenarios.

5.2.1 Problem Formulation

Consider a $\langle \mathcal{S}, \mathcal{A}, P, R, N, B \rangle$ environment consisting of N projects, each characterized by its unique state, action, and the resulting reward. Specifically, the state of the system is given by $\mathbf{s} = (s_1, ..., s_N) \in \mathcal{S}$, where each project is represented as s_i, an element from the finite state space $\mathcal{S}_i$, $i = 1, ..., N$. Analogously, the actions taken in each project are denoted as elements a_i belonging to the compact action space $\mathcal{A}_i$, and the complete action vector is denoted as $\mathbf{a} = (a_1, ..., a_N) \in \mathcal{A}$. The rewards obtained from these actions are encapsulated as elements r_i in the reward vector $\mathbf{r}$. The cost associated with each action a_i is expressed as $c(a_i)$, and the cumulative cost for all actions is given by $C(\mathbf{a}) = \sum_i c(a_i)$.

The system dynamics are governed by a transition probability function $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ which specifies the probabilities of transitioning to new states given particular state and action vector. Given the values of actions, P has a product form. A discount factor $\gamma \in (0, 1)$ is used to balance immediate and future rewards.

The long-term discounted reward can be expressed through the Bellman Value function $V(\mathbf{s})$, which is the expected sum of discounted rewards accumulated over time, starting from the state s and satisfying the Bellman dynamic programming equation [66]:

$$V(\mathbf{s}) = \max_{\mathbf{a}\in\mathcal{A},\, C(\mathbf{a})=B} \left[\sum_{i=1}^{N} r_i(s_i, a_i) + \gamma \mathbb{E}[V(\mathbf{s}') \mid \mathbf{s}, \mathbf{a}] \right]. \tag{5.4}$$

5.2.2 Lagrangian Decomposition for Continuous Actions

The complexity of the problem comes primarily from the constraint imposed on the actions, which are dictated by a common pool of resources. Specifically, each project must select a continuous action $a_i \in [0, a_i^{\max}]$. Their total activation cost, represented by the total cost $C(\mathbf{a})$, consumes from the available resources B. This shared resource pool constraint means that actions across projects are inherently coupled, which significantly increases the complexity of the decision space as the number of projects increases. The exponential growth in decision space complexity due to this coupling highlights the challenge of resource allocation and emphasizes the need for efficient use of the shared resource pool [67].

To manage this complexity, [43] relaxed the value function using a Lagrange multiplier λ. This transforms the original problem into a Lagrangian form, seeking to maximize the value function while adhering to the resource constraint:

$$J(\mathbf{s}, \lambda) = \max_{\mathbf{a}\in\mathbf{A}} \left[\sum_{i=1}^{N} r_i(s_i, a_i) + \lambda \left(B - \sum_{i=1}^{N} c(a_i) \right) + \gamma \mathbb{E}[J(\mathbf{s}', \lambda) \mid \mathbf{s}, \mathbf{a}] \right]. \tag{5.5}$$

Here, λ is the Lagrange multiplier associated with the resource constraint B. By adjusting λ, we effectively balance the immediate cost of actions against their long-term rewards, allowing for a decoupling of the projects' decisions.

If we assume the additive structure of the value function with respect to the projects of the weakly coupled MDP, the Equation (5.5) can be rewritten as:

$$J(\mathbf{s}, \lambda) = \frac{\lambda B}{1-\gamma} + \sum_{i=1}^{N} \max_{a_i \in A_i} Q_i(s_i, a_i, \lambda), \tag{5.6}$$

where

$$Q_i(s_i, a_i, \lambda) = r_i(s_i, a_i) - \lambda c(a_i) + \gamma \sum_{s_i'} P(s_i, a_i, s_i') \max_{a_i' \in A_i} Q_i(s_i', a', \lambda). \tag{5.7}$$

In this decoupled framework, the Lagrange multiplier λ is key in determining the optimal policy for each project. Under the budget constraint B, λ acts as a trade-off parameter by introducing a penalty term $\lambda c(a_i)$ for the actions taken. A higher

λ parameter places more emphasis on minimizing the cost (i.e., staying within the resource limit B), while a lower λ value shifts the focus towards maximizing rewards with less emphasis on the cost implementations. As λ rises, the preferred policy for each project will increasingly prefer actions that offer the highest "value-to-cost" ratio. Thus, the function (5.6) is a measure of the total expected reward, adjusted for the cost of the actions taken under that policy.

To balance the expected rewards with the cost of actions, we need to find λ^* such that

$$\lambda^* = \arg\min_{\lambda} J(\mathbf{s}, \lambda). \tag{5.8}$$

This term is defined as the best trade-off between maximizing rewards and minimizing the cost of actions. It is at this point that the policy aligns with the discounted time-averaged resource constraints, ensuring that the actions selected are not only rewarding but also resource-efficient.

Then, in a continuous action framework, at each time step t we aim to solve the following Knapsack optimization problem:

$$\max_{\mathbf{a}\in\mathbf{A}} \quad \sum_{i=1}^{N} Q_i(s_i(t), a_i, \lambda^*) \qquad s.t. \quad \sum_{i=1}^{N} c_i(a_i) = B. \tag{5.9}$$

In the LPCA algorithm, described in detail next, we interpolate the curve of the Q-values $Q(s, a, \lambda)$ as functions of the Lagrange multiplier λ through a neural network. This curve is a convex function with respect to λ ([43]), making the minimization of (5.6) a simple one-dimensional convex optimization problem once the neural network is trained. For the optimization (5.9) we explore two approaches as outlined in Sections 5.2.4 and 5.2.5.

5.2.3 LPCA Algorithm

The core methodology of the LPCA algorithm involves a two-timescale process centered around learning and optimization. Initially, LPCA focuses on training a neural network to accurately approximate the Q-values as defined in Equation (5.7). This process involves learning the balance between immediate rewards, action costs, and future rewards based on the transition dynamics of the system. Once the neural network is effectively trained for the current coupled state s, LPCA computes the value function $J(\mathbf{s}, \lambda)$ as described in Equation (5.6). The objective is to determine the optimal Lagrange multiplier λ^* that minimizes $J(\mathbf{s}, \lambda)$ as formulated in Equation (5.8). Finally, LPCA addresses the optimization problem set out in Equation (5.9) through two possible methods: a differential evolution optimizer, described in Algorithm 8, or a greedy optimizer presented in Algorithm 9.

The general training process of LPCA, as outlined in Algorithm 5, is a key aspect of our approach. The algorithm begins by using a policy dictionary to interact with the environment. This dictionary is a mapping of states to actions, where each

Algorithm 5 LPCA Training Process

```
Require: Environment, $N_{\text{iter}}$, Update frequency $N$, Batch size $M$, Policy method
Ensure: Train LPCA Model, Update Policy Dictionary
Initialize Q-value neural network, policy dictionary, experience memory
for iteration $= 1$ to $N_{\text{iter}}$ do
    Select and execute action $\mathbf{a}$, store $(\mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}', \text{done})$
    if memory $\geq M$ then
        Update Q-values with mini-batch of $M$ (Algorithm 6)
    end if
    if iteration $\mod N = 0$ then
        Update policy with Differential Evolution or Greedy (Algorithm 7)
    end if
end for
```

state corresponds to a unique action vector. During each interaction, an action is selected based on the current policy, and the environment responds accordingly. The response, including the state transition and reward information, is stored as a transition sample. Notably, each process of the weakly coupled MDP is treated individually, with the transition sample from each project recorded separately in a memory buffer. This memory serves as a repository for experiences, which are later used to update the neural network that approximates Q-values.

The training of the neural network, as detailed in Algorithm 6, is central to learning the Q-values from Equation (5.7) associated with state transitions (s, a, r, s') across a range of test λ values. These test values are selected as a random subset, which encompasses a discretized set of λ values in the range of a problem-dependent $[-\lambda_{\text{max}}, \lambda_{\text{max}}]$, using 1000 discretization.

During each iteration of the training process, the algorithm samples a batch of experiences from the memory. Each experience comprises the current state s, the action taken a, the reward received r, the subsequent state s', and a boolean flag indicating the terminal status of s' for a given individual project. For each experience, the algorithm computes the target Q-values for the state-action pair (s, a) using a random subset of λ values. This step involves evaluating the Q-value function for different levels of resource utilization and cost. The computation of the target Q-values $Q_{target}(s, a, \lambda)$ utilizes a target network, which is a lagged version of the primary neural network, to provide stable targets for learning [68].

Through this training process, the LPCA algorithm efficiently learns the Q-values for various state transitions under different levels of resource constraints, as dictated by the varying λ values.

Once we have trained the neural network to generate accurate approximations of the Equation (5.7), we proceed with Algorithm 7 to compute the value function $J(\mathbf{s}, \lambda)$ for a given state $\mathbf{s}$ as in Equation (5.6). This calculation involves evaluating $\sum_{i=1}^{N} \max_{a_i} Q(s_i, a_i, \lambda)$ for each λ within a discretized set.

Algorithm 6 Update Q-values in LPCA Neural Network Model

```
for each random sample in memory do
    Extract $s$, $a$, $r$, $s'$, $done$ from sample
    $Q \leftarrow$ Calculate target Q-values for $s$ and $a$ using a subset of $\lambda$
    $V_{expected} \leftarrow$ Calculate expected value functions for $s'$ using target network for each $\lambda$
    if done then
        $Q_{target}(s, a, \lambda) \leftarrow r(s) - \lambda c(a)$
    else
        $Q_{target}(s, a, \lambda) \leftarrow r(s) - \lambda c(a) + \gamma \cdot V_{expected}$
    end if
    Perform a gradient descent step on $(Q_{target}(s, a, \lambda) - Q(s, a, \lambda))^2$ to update network weights
end for
Perform soft-update on target network weights $\theta' \leftarrow \theta\tau + (1 - \tau)\theta'$
```

Once this term is computed, finding the optimal λ^* is a one-dimensional convex optimization problem, as shown in Equation (5.8).

This recoupling of the individual s_i states into a coupled s state is critical for synthesizing the individual decisions across different MDPs into a coherent policy.

While the initial decoupling in LPCA helps simplify the problem and allows for faster and more accurate neural network training - as each subproblem becomes more manageable - the recoupling phase introduces significant computational complexity. The size of the state space, which contains all possible combinations of individual MDP states, grows exponentially with the number of MDPs involved, as discussed in Section 2.2.2. This exponential growth poses a major challenge, as it drastically increases the complexity required to optimize this large state space. Moreover, computation of a complete policy dictionary becomes exponentially more intensive as the number of MDPs increases. One proposed solution, given a large state space, is to compute the policy dictionary online, only for the visited states, in case the inference phase is shorter than the number of coupled states, making an upper bound for the computational complexity to the number of visited states.

A key technical contribution of our work is how we explore the action space to solve the knapsack problem described in Equation (5.9). We propose two different strategies to explore this action space in order to make the best use of the available resources and select the best action based on our Q-value estimates. The first strategy, presented in Section 5.2.4, is an evolutionary algorithm (LPCA-DE). It uses mechanisms similar to natural selection to iteratively search for the optimal solution, effectively avoiding local minima by exploring a wider range of solutions.

The second strategy, presented in Section 5.2.5, is a greedy algorithm (LPCA-Greedy). It focuses on choosing the action based on the gradient of the Q-values

Algorithm 7 Computation of Lagrange term λ^*

```
Require: method
Ensure: Updated policy dictionary $\pi(\mathrm{s})$
function PolicyDictUpdate(method)
for all s $\in$ S do
    $Q \leftarrow$ Zero Matrix of size [n_lambda, N]
    for $i \in 1 : N$ do
        $Q[:, i] \leftarrow \max_{a_i} Q(s_i, a_i, \lambda), \forall\lambda$
    end for
    $J(\mathrm{s}, \lambda) \leftarrow$ Compute value functions as (5.6)
    $\lambda^*(\mathrm{s}) \leftarrow \arg\min_\lambda J(\mathbf{s}, \lambda)$
    if method = Evolution then
        $\mathbf{a}^* \leftarrow$ DifferentialEvolution(s, $\lambda^*(\mathrm{s}), a_{\max}$)
    else
        $\mathbf{a}^* \leftarrow$ Greedy(s, $\lambda^*(\mathrm{s}), a_{\max}, \delta$)
    end if
    $\pi(\mathrm{s}) \leftarrow \mathbf{a}^*$
end for
end function
```

with respect to the actions for each project, selecting the action that promises the highest increase in the Q-value per unit of resource expended. This method is simpler and faster, and helps to quickly identify actions that increase payoff.

The main goal of employing the differential evolution and greedy approaches within the LPCA framework is to effectively navigate this expansive action space. These methods are specifically chosen over gradient-based optimization techniques, which, while powerful in certain contexts, often fail in this setting due to their propensity to become trapped in local optima.

5.2.4 Differential Evolution Optimization (LPCA-DE)

The first method, explained in Algorithm 8, employs a differential evolutionary algorithm, renowned for its effectiveness in identifying global optima and circumventing local optima traps. This method is particularly adept at exploring the search space comprehensively [69].

Evolutionary algorithms (EAs) are a class of stochastic, population-based optimization algorithms inspired by biological evolution. These algorithms operate on the principle of "survival of the fittest", where a population of potential solutions evolves over generations towards an optimal solution.

The general process of an evolutionary algorithm involves the initialization of a population of solutions followed by the application of genetic operators such as selection, crossover, and mutation. The selection process favors individuals (solutions) with higher fitness scores to pass their genes (solution attributes) to the next generation. Crossover and mutation introduce variability and new traits into the offspring, promoting exploration of the solution space. The algorithm iterates through these steps until a termination criterion is met, such as reaching a maximum number of generations or achieving a satisfactory fitness level.

The mathematical formulation of an evolutionary algorithm typically involves defining a fitness function $f(x)$, which evaluates the quality of each solution x in the population. The objective is to maximize (or minimize) this fitness function. The evolutionary process can be described by the following generic steps:

1. **Initialization**: Generate an initial population P_0 of solutions randomly or based on heuristic information.
2. **Evaluation**: Compute the fitness $f(x)$ for each solution x in the current population.
3. **Iterate:**
   a) **Mutation**: add weighted difference between two solutions and a third one
   $$x_{mutant} = x_{target} + T(x_B - x_C)$$
   b) **Crossover**: combine mutated solutions with the current solution to generate a trial solution. For a given crossover rate R and trial vector t, each of its elements will be defined as
   $$t_i = \begin{cases} m_i & \text{if rand}(i) \leq R \text{ or } i = i_{rand} \\ a_i & \text{otherwise} \end{cases}$$
   c) **Selection**: Choose the solution with the best fitness to pass to the next generation.
4. **Convergence**: Repeat until the population converges to the optimal solution.

The actions obtained through this method are solutions to the relaxed version of the problem, where the resource constraint is averaged over time. This leads to situations where the total cost of the chosen action vector temporarily exceeds or falls below the resource constraint B. Thus, a critical aspect of this approach is the integration of a penalty mechanism to ensure that the choice of actions remains within the resource constraints of the original problem. Actions that result in resource usage beyond the available limit are subject to a significant penalty. Without this penalty, the optimizer may select actions that exceed the resource limit, leading to infeasible solutions.

On the other hand, it's not uncommon for an action to use less than the available resources. Given the $\lambda c(a)$ term in Equation (5.7), the derived optimal policy tends to be cost-effective. However, it may not always coincide with the optimal policy

Algorithm 8 Action Selection through Differential Evolution Optimization

```
Require: State vector s, fixed Lagrange multiplier $\lambda_{\text{fix}}$, maximum action $a_{\max}$
Ensure: Optimal actions maximizing Q-values under resource constraints
function DifferentialEvolution(s, $\lambda_{\text{fix}}$, $a_{\max}$)
Bounds $\leftarrow [0, a_{\max}]$
function ObjectiveFunction(a, s, $\lambda^*$)
$Q_{\text{total}} \leftarrow \sum_{i=1}^{N} Q(s_i, a_i, \lambda^*)$
$C_{\text{total}} \leftarrow \sum_{i=1}^{N} C(s_i, a_i)$
if $C_{\text{total}} > B$ then
    Penalty $\leftarrow$ Large constant value
    $Q_{\text{total}} \leftarrow Q_{\text{total}} -$ Penalty
else if $C_{\text{total}} < B$ then
    Penalty $\leftarrow B - C_{\text{total}}$
    $Q_{\text{total}} \leftarrow Q_{\text{total}} -$ Penalty
end if
return $-Q_{\text{total}}$
end function
a* $\leftarrow$ Apply Differential Evolution optimization with $(ObjectiveFunction, Bounds)$

return a*
end function
```

of the original constrained problem (5.4), leading to potential underutilization of resources.

To address this, we introduce a secondary optional penalty, equal to the amount of unused resources, into the differential evolution optimization problem. This modification guides the optimizer toward actions that maximize resource utilization, ensuring that the algorithm not only pursues cost-effective solutions, but also fully utilizes the available resources.

5.2.5 Greedy Optimization Strategy (LPCA-Greedy)

The greedy optimization strategy, as described in the Algorithm 9, develops through a gradual iterative process. This method evaluates the gradient of the Q-values in relation to the actions for each project, and then directs resources to the project with the highest gradient. This process is repeated until the pool of available resources is completely utilized.

Our approach reflects a strategic focus on maximizing the use of resources, optimally allocating resources to projects that have the greatest potential for increasing the value of quality per unit of resource consumed. Unlike the differential evolution approach, which searches for the optimal policy before making adjustments to match resource availability, the greedy method operates on the principle of ensuring total resource utilization. It achieves this by systematically allocating

Algorithm 9 Greedy Action Selection for Continuous MDP

```
Require: State s, $\lambda_{\text{fix}}$, max action $a_{\max}$, increment $\delta$
Ensure: Optimal actions maximizing Q-values, maximum action $a_{\max}$
function Greedy($\mathbf{s}, \lambda_{\text{fix}}, a_{\max}, \delta$)
Initialize action vector a to zeros, $B_{\text{remaining}} = B$
while $B_{\text{remaining}} > 0$ do
    $i \leftarrow \arg\max_i \frac{\partial Q}{\partial a_i}$
    $a_i \leftarrow a_i + \delta$, ensure $a_i \leq a_{\max}$
    $B_{\text{remaining}} \leftarrow B - \sum_{i=1}^{N} c(s_i, a_i)$
end while
return a
end function
```

resources in such a way as to optimize the incremental benefit gained from each project.

5.3 Experimental Setup

Our LPCA models are intricately designed to tackle multi-armed bandit problems characterized by continuous actions. These types of problems have traditionally been addressed by techniques such as actor-critic models.

Actor-critic models employ a dual neural network structure: the actor network, which suggests an action vector $\mathbf{a}$ given a complete state vector $\mathbf{s}$, and the critic network, which is tasked with evaluating the actor network's performance $Q(\mathbf{s}, \mathbf{a})$. The actor embodies the agent's policy, seeking to generate actions that maximize expected returns and refining its strategy based on the critic's insights. Meanwhile, the critic's role is to predict future rewards from any given state by evaluating the actions generated by the actor.

Learning in these models evolve in two concurrent phases: the critic refines its value function for more accurate predictions of state or state-action pair values, while the actor modifies its policy using the critic's evaluations to improve decision efficiency.

A notable example of such models is the Deep Deterministic Policy Gradient (DDPG) [70], which combines principles from DQN (Deep Q-Networks) and deterministic policy gradient techniques. DDPG is characterized by three main components: the actor network, which directly outputs the optimal action for each state; the critic network, which evaluates the actor's output by estimating its Q-value; and target networks for both actor and critic, which reflect the double DQN [68] technique to enhance stability. The critic's learning process involves decrease the gap between its Q-value predictions and the target Q-values, which are derived from observed rewards and the target critic's predicted Q-values of the upcoming state. At the same time, the actor updates its policy by using the gradient of the critic's output with respect to the actions to guide policy improvement.

However, DDPG and actor-critic models in general traditionally generate actions that are not bounded by resource constraints and are only limited by the action space. To address this shortcoming, OptLayer, introduced by [71], modifies the actor network by incorporating an additional layer that formulates the convex optimization problem of Equation (5.10):

$$\begin{aligned} \text{minimize} \quad & \|Wx + b - y\|_2^2 \\ \text{subject to} \quad & \sum y \leq B \\ & 0 \leq y \leq a_{\max} \\ & W_{\text{tilde}} = W \end{aligned} \tag{5.10}$$

Where W and b are the learnable parameters of the new layer, x is the output of the previous layer (originally the actions that the Actor Network would produce), and y are the new outputs of the Actor Network, the constrained actions that satisfy the total resource constraint and the bound in the action space. This layer aims to reconcile the actions proposed by the actor with the prevailing resource constraints, ensuring that the actions not only comply with the space boundaries, but also with the overall resource constraints.

We will use this model, together with the optimal heuristic from the Whittle indices computed in Section 5.1.1, to evaluate the performance of our LPCA models in several continuous actions weakly coupled problems.

5.3.1 Description of Test Environments for Continuous Models

To evaluate the performance of our LPCA algorithms against theoretical Whittle indices and the DDPG algorithm augmented with OptLayer, we will use four different environments for testing: "Type A" and "Type B" from [72], both adapted for continuous actions, as well as an admission control and a speed scaling problem.

5.3.1.1 Type A and Type B Environments

"Type A" and "Type B" environments are characterized by having two states per arm, with their reward and cost functions simply defined in terms of the state of the MDP and the action performed, respectively:

$$R(s, a) = s \qquad C(s, a) = a$$

A defining feature of these environments is their transition probabilities. For "Type A", we have adapted the original probabilities for three discrete actions ($a = \{0, 1, 2\}$) from [72] to a continuous action spectrum $a \in [0, a_{\max}]$, with

$a_{\max} = 2$, resulting in:

$$P_A(s,a) = \begin{pmatrix} 0.02a^2 - 0.09a + 0.8 & -0.02a^2 + 0.09a + 0.2 \\ 0.75e^{-0.947a} & 1 - 0.75e^{-0.947a} \end{pmatrix}$$

and for "Type B":

$$P_B(s,a) = \begin{pmatrix} 0.95e^{-2.235a} & 1 - 0.95e^{-2.235a} \\ 0.3347e^{-1.609a} & 1 - 0.3347e^{-1.609a} \end{pmatrix}$$

In "Type A", the transition probability to stay in state 1 increases with the action at a faster rate than the probability to go from state 0 to state 1. This scenario leads to a greater "return on investment" when allocating resources to MDPs in state 1 than in state 0.

"Type B" presents the opposite dynamic, with the transition probabilities of state 0 to state 1 increasing at a faster rate than the probability of remaining in state 1, suggesting an increased "return on investment" from prioritizing MDPs in state 0.

5.3.1.2 Admission Control Environment

Our third environment will be the admission control problem, where we consider each MDP as a processor managing incoming jobs with a maximum queue length $s_{\max} = 4$. Here, for each MDP i, the arrival rate $\alpha_i(a_i) = \alpha_{\max}\frac{2-a}{2}$ can be modulated by the agent's actions, where $\alpha_{\max} = 2$ is the maximum arrival rate and $\mu = 1$ is the fixed departure rate. The action parameter a determines the rejection rate of incoming jobs: an action $a = 0$ leads to the acceptance of all jobs, while $a = 2$ leads to their complete rejection. This scenario, originally set in continuous time, requires a discretization to be compatible with our algorithms designed for discrete time models.

This adjustment introduces a normalization factor $\nu = \alpha_{\max} + \mu$. The discount factor, which is used to reduce future rewards and allow convergence in long-term rewards (as seen in Section 2.1.2), is defined in continuous-time models as β, and discounts rewards as $R = \sum_{t=0}^{T} r(t)e^{-\beta t}$. Given the current normalization factor ν, we can relate this continuous discount factor to the discrete discount γ as $\gamma = \frac{\nu}{\nu+\beta}$.

The transition probabilities for this time-discretized model are expressed as:

$$P(a) = \begin{pmatrix} 1 - \frac{\alpha(a)}{\nu} & \frac{\alpha(a)}{\nu} & 0 & 0 \\ \frac{\mu}{\nu} & 1 - \frac{\alpha(a)+\mu}{\nu} & \frac{\alpha(a)}{\nu} & 0 \\ 0 & \frac{\mu}{\nu} & 1 - \frac{\alpha(a)+\mu}{\nu} & \frac{\alpha(a)}{\nu} \\ 0 & 0 & \frac{\mu}{\nu} & 1 - \frac{\mu}{\nu} \end{pmatrix}$$

The reward function of the problem is structured to penalize higher states, especially the final state $s_{\max}$, by including a "rejection cost" C_r. This leads to the definition:

$$R(s,a) = \frac{s_{\max} - s - C_r(s,a)}{\nu + \beta} = \begin{cases} \frac{s_{\max} - s - \alpha_{\max}\frac{a}{2}}{\nu+\beta} & \text{if } s < s_{\max} \\ \frac{-\alpha_{\max} - 18}{\nu+\beta} & \text{if } s = s_{\max} \end{cases}$$

Furthermore, the activation cost function is defined to reflect the cost of actions in non-terminal states and to nullify it for the terminal state $s_{\max}$, due to the lack of action dependence in the transition probabilities at this state:

$$C(s,a) = \begin{cases} \frac{\frac{a^2}{2} + 2a}{\nu+\beta} & \text{if } s < s_{\max} \\ 0 & \text{if } s = s_{\max} \end{cases}$$

To avoid a scenario where the final state $s_{\max}$ might appear unreasonably advantageous due to the absence of activation costs, we significantly increase $C_r(s_{\max}, \cdot) = \lambda_{\max} + 18$ for $s_{\max}$, emphasizing the critical nature of this state.

5.3.1.3 Speed Scaling Environment

The final environment to be explored in our study is the speed scaling problem. Similar to the admission control environment, each MDP in this setting acts as a processor for incoming jobs. However, unlike the previous case, here the agent has the authority to control the departure rate $\mu_i = \sqrt{a_i}$, while the arrival rate is maintained at a fixed value $\alpha = 0.9$, and the system is capped at a maximum state $s_{\max} = 6$.

Originally formulated in continuous time, the transition to a discretized framework involves employing a normalization factor $\nu = \alpha + \mu_{\max}$ and converting the continuous discount factor β to its discrete counterpart γ via $\gamma = \frac{\nu}{\nu+\beta}$. This adjustment results in the following transition probability matrix:

$$P(a) = \begin{pmatrix} 1 - \frac{\alpha}{\nu} & \frac{\alpha}{\nu} & 0 & 0 & 0 & 0 \\ \frac{\mu_a}{\nu} & 1 - \frac{\alpha+\mu_a}{\nu} & \frac{\alpha}{\nu} & 0 & 0 & 0 \\ 0 & \frac{\mu_a}{\nu} & 1 - \frac{\alpha+\mu_a}{\nu} & \frac{\alpha}{\nu} & 0 & 0 \\ 0 & 0 & \frac{\mu_a}{\nu} & 1 - \frac{\alpha+\mu_a}{\nu} & \frac{\alpha}{\nu} & 0 \\ 0 & 0 & 0 & \frac{\mu_a}{\nu} & 1 - \frac{\alpha+\mu_a}{\nu} & \frac{\alpha}{\nu} \\ 0 & 0 & 0 & 0 & \frac{\mu_a}{\nu} & 1 - \frac{\mu_a}{\nu} \end{pmatrix}$$

In this setting, the reward function penalizes the higher states. Contrary to the previous environment, the only time a rejection occurs is in the final state, with its corresponding penalty. The reward function is defined as:

$$R(s) = \frac{-s}{\nu + \beta} + \frac{C_r}{\nu + \beta} = \begin{cases} \frac{-s}{\nu+\beta} & \text{if } s < s_{\max} \\ \frac{-s_{\max} - 10}{\nu+\beta} & \text{if } s = s_{\max} \end{cases}$$

Including an additional rejection cost $C_r = -10$ in the final state increases the penalty for reaching that state.

The first state, $s = 0$, is considered uncontrollable, since job departures are infeasible in the absence of jobs. Therefore, the cost function is

$$C(s,a) = \begin{cases} 0 & \text{if } s = 0 \\ \frac{a}{\nu+\beta} & \text{if } s > 0 \end{cases}$$

5.3.2 Evaluation Metrics and Benchmarks

Given the differences between LPCA's index heuristics and the traditional Whittle index for continuous actions, our comparative analysis will focus exclusively on averaged discounted rewards obtained by each model's policy. This approach is in accordance with the methodology outlined in Section 4.3.2 and ensures a consistent framework for evaluating algorithm performance.

During each evaluation epoch, we examine the performance of various algorithms, including LPCA-DE, LPCA-Greedy, DDPG equipped with OptLayer, and the theoretical continuous action Whittle indices, over the previously discussed environments. This evaluation process involves performing $N_e = 10$ evaluation runs, each going through 500 randomly selected initial states. For each state, we compute the total discounted reward over $n_{iter} = 50$ iterations, following the policy by each algorithm, resulting in an individual reward R_i. These rewards are then averaged across all initial states to derive R, a process that is repeated for each evaluation run to assemble a vector $\mathbf{R}$ of averaged rewards.

To encapsulate the performance of each algorithm during epoch t, we compute both the mean of $\mathbf{R}$ and its confidence interval, the latter denoted by the standard deviation $\sigma(\mathbf{R}(t))$.

Table 5.1 provide a summary of the performance of each algorithm in terms of the relative error of the value function $V_{\pi(t)}$ in the last iteration with respect to the theoretical Whittle index policy, defined as:

$$\text{Relative Error} = \frac{|V_{\pi(t)} - V_{\text{Whittle}}|}{|V_{\text{Whittle}}|}.$$

5.3.3 Detailed Analysis of Experimental Results

In this section, we present the numerical results obtained from the implementation of our models in the previously detailed environments: Type A, Type B, Admission Control, and Speed Scaling. Each environment is designed to rigorously test the effectiveness of our algorithms-LPCA variants and DDPG with OptLayer-against the theoretical continuous Whittle indices, under different operational settings.

A single discrete discount factor, $\gamma = 0.9$, is used across all scenarios to ensure comparability. For environments originally formulated in continuous time, such as the Admission Control and Speed Scaling problems, the conversion to a discrete time framework requires the definition of a continuous discount factor, β. This factor is computed from the discrete γ and the maximum arrival rates λ and departure rates μ, using the formula

$$\beta = \frac{\lambda + \mu}{\gamma} - (\lambda + \mu).$$

To evaluate performance across different levels of complexity and resource allocation, each environment is tested with two different configurations:

- With $N = 4$ arms, where the available resources are just enough to fully activate one arm, allowing us to evaluate the effectiveness of the algorithms in distributing resources.
- With $N = 6$ arms, where resources are sufficient to fully activate two arms, thus increasing decision complexity and testing the algorithms' ability to effectively manage more substantial resources and processes.

The progression through the environments is structured to escalate in difficulty, not only in terms of requiring more nuanced policies for effective management, but also by expanding the state space, which challenges the computational robustness and adaptability of the algorithms.

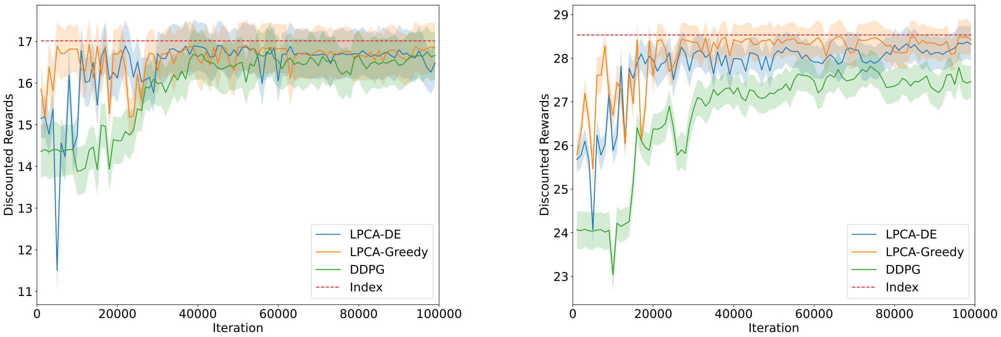
Type A Environment.

In the Type A environment, our analysis is structured around two setups designed to test the algorithms under resource-constrained conditions: the first with $N = 4$ arms and a total resource budget of $B = 2$, and the second with $N = 6$ arms and a resource budget of $B = 4$. These setups imply that the cumulative action across all MDPs satisfies the conditions $\sum_{i=1}^{N} a_i = 2$ and $\sum_{i=1}^{N} a_i = 4$, respectively. Given the cost function of this environment, these configurations severely limit the ability of the MDPs to operate at full capacity, thereby imposing a strategic requirement for sensible resource management and allocation.

In the first configuration with $N = 4$, as shown in Figure 5.2 left, both LPCA variants, LPCA-DE and LPCA-Greedy, show comparable performance. In particular, from about iteration 30000, their results are close to the theoretical continuous Whittle index, indicating a robust adherence to optimal policy guidelines. In contrast, DDPG initially lags behind during the first 40000 iterations, but eventually converges to a similar performance level as the LPCA algorithms.

Moving to the more complex $N = 6$ scenario shown in Figure 5.2 right, the differences between the algorithms become more pronounced. In this setting, LPCA-Greedy slightly outperforms LPCA-DE, converging to the performance level of the Whittle index and suggesting more efficient resource utilization and policy execution. DDPG, which previously matched the LPCA algorithms in the $N = 4$ setting, now exhibits significantly weaker performance. This decline highlights potential scalability issues with DDPG when faced with an increased number of arms and narrower resource constraints.

Figure 5.2 Performance comparison of LPCA-DE, LPCA-Greedy, DDPG with Opt-Layer, and Whittle indices for the Type A environment with $N = 4$ and $B = 2$ (left) and $N = 6$ and $B = 4$ (right)



Type B Environment. In the Type B environment, we continue the evaluation under the same configurations as previously described: with $N = 4$ arms and a resource budget of $B = 2$, and thereafter with $N = 6$ arms and a resource budget of $B = 4$. These setups require that the collective actions across all MDPs satisfy the constraints $\sum_{i=1}^{N} a_i = 2$ and $\sum_{i=1}^{N} a_i = 4$, respectively.

For the initial setup of $N = 4$, as shown in Figure 5.3 left, both LPCA algorithms-LPCA-DE and LPCA-Greedy-exhibit a remarkable ability to closely match the performance level of the theoretical Whittle index policy. In contrast, DDPG exhibits a more erratic trajectory; its performance fluctuates, aligning with the LPCAs in some iterations while falling behind significantly in others. This variability highlights potential challenges in DDPG's adaptability or stability in consistently learning optimal policies in this environment.

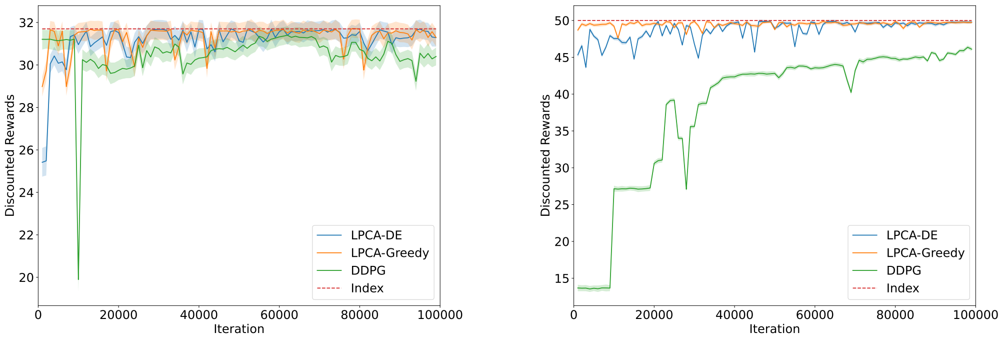
The complexity and challenges escalate with the expansion to $N = 6$ arms and $B = 4$, as shown in Figure 5.3 right. In this more challenging scenario, DDPG's performance continues to deteriorate, significantly underperforming the Whittle index policy and exhibiting significant deviation from optimal policies even as it progresses through the training phases. This marked decline highlights scalability issues or inefficiencies within DDPG's learning mechanism when faced with increased complexity and tighter resource constraints.

In contrast, both LPCA variants maintain their performance and converge close to the Whittle index benchmark. In particular, the LPCA greedy variant is characterized by excellent stability and robustness throughout the training process. Remarkably, it achieves a high level of performance from the outset, demonstrating its efficiency in quickly learning and adhering to highly effective policies with minimal transition data requirements.

Admission Control Environment.

The admission control environment presents a more complex scenario than the Type A and Type B environments, with a larger state space per arm and more

Figure 5.3 Performance comparison of LPCA-DE, LPCA-Greedy, DDPG with OptLayer, and Whittle indices for the Type B environment with $N = 4$ and $B = 2$ (left) and $N = 6$ and $B = 4$ (right)



complex dynamics.

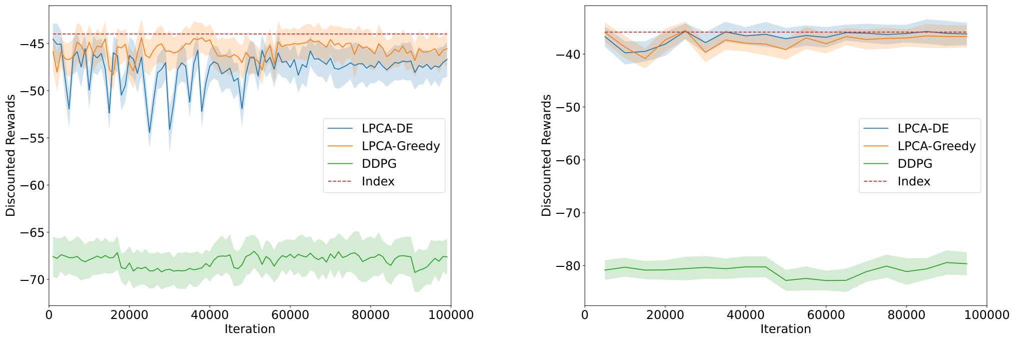
For the admission control environment, we keep the same resource constraints as in the previous settings, which translate to $\sum_{i=1}^{N} a_i = 2$ and $\sum_{i=1}^{N} a_i = 4$. However, due to the different cost function associated with this problem, these constraints are adjusted to $N = 4$ with $B = 1.8$ and $N = 6$ with $B = 3.6$. Despite these modifications, the core challenge of resource allocation remains critical.

Looking at the first configuration of $N = 4$ arms with a resource budget of $B = 1.8$, shown in Figure 5.4 left, we observe a strong divergence in the performance results of the evaluated algorithms. In particular, DDPG shows a pronounced inability to learn an effective policy; its performance remains well below the theoretical Whittle index threshold throughout the training period, and it significantly underperforms compared to both LPCA versions. This inability to adapt or learn efficiently in more complex scenarios underscores potential limitations in the utility of DDPG with OptLayer in certain contexts.

Conversely, both versions of LPCA - LPCA-DE and LPCA-Greedy - show a much closer approximation to the Whittle index performance, with LPCA-Greedy in particular excelling due to its strategic incremental action assignment. This method allows it to allocate resources judiciously without the need for exhaustive action exploration, thus maintaining stability and consistency in performance. LPCA-DE, while generally effective, experiences fluctuations in the first half of the training process, especially in the early stages, with several negative performance spikes, indicative of its broader exploratory approach in the action space.

In the second setting, shown in Figure 5.4 right, the admission control environment shows trends consistent with our earlier observations: DDPG continues to exhibit limitations and adheres to a suboptimal policy throughout the evaluation period. This performance stagnation highlights DDPG’s difficulty in effectively scaling with an increased number of arms and more complex resource allocation requirements.

Figure 5.4 Performance comparison of LPCA-DE, LPCA-Greedy, DDPG with OptLayer, and Whittle indices for the Admission Control environment with $N = 4$ and $B = 1.8$ (left) and $N = 6$ and $B = 3.6$ (right)



In contrast, the LPCA algorithms-especially LPCA-DE-demonstrate significant adaptability and robustness under these increased conditions. Both LPCA-DE and LPCA-Greedy closely approximate Whittle index policy performance, with LPCA-DE showing particular strength in this regard. Their consistent performance confirms the algorithms' potential to effectively manage larger systems, making them suitable candidates for complex real-world applications where resource optimization is critical.

Speed Scaling Environment.

The speed scaling problem stands out as the most complex, due to its larger state space per arm.

For this analysis, we will use the activation conditions as before, which, given the cost function of the problem, lead to the settings $N = 4$ with $B = 0.774$, the total resources required to fully activate one arm, and $N = 6$ with $B = 1.5478$, the resources required to fully activate two arms.

In the first setting, shown in Figure 5.5, DDPG shows initial learning activity within the first 20000 iterations, but soon reaches a plateau and eventually locks into a significantly suboptimal policy. This behavior mirrors the difficulties observed in the admission control problem, where DDPG's performance not only fails to reach the theoretical Whittle index policy threshold, but also falls behind the LPCA variants significantly.

In contrast, both LPCA-DE and LPCA-Greedy show remarkable performance in this environment. After about 30000 iterations, their performance trajectories closely match, each converging toward the Whittle index policy performance. Despite this convergence, there remains a noticeable gap between their performance and the optimal policy level, which is larger than that observed in previous environments. This gap underscores the increased difficulty of the speed scaling problem, where even the more robust LPCA algorithms struggle to fully meet the performance benchmark set by the Whittle index.

Figure 5.5 Performance comparison of LPCA-DE, LPCA-Greedy, DDPG with OptLayer, and Whittle indices for the Speed Scaling environment with $N = 4$ and $B = 0.7739$

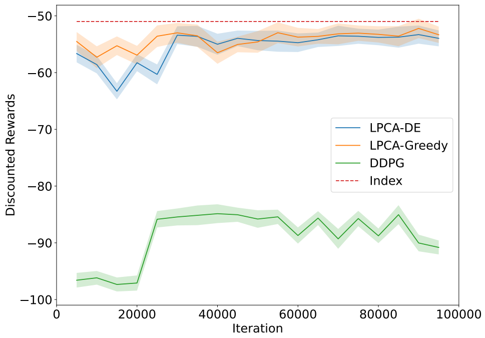


Table 5.1: Relative Error in Algorithm Performance at Final Iteration for Continuous Actions Problems in %

	LPCA DE	LPCA Greedy	DDPG+OptLayer
Type A 4 arms	3.1000 %	**0.9804** %	2.0595 %
Type A 6 arms	0.7402 %	**0.3946** %	3.7510 %
Type B 4 arms	1.3057 %	**1.2772** %	4.1078 %
Type B 6 arms	**0.4408** %	0.5285 %	7.8006 %
Admission Control 4 arms	6.1352 %	**3.5535** %	53.6963 %
Admission Control 6 arms	**0.9733** %	2.3220 %	122.3079 %
Speed Scaling 4 arms	5.8284 %	**4.5809**%	78.0848 %
Speed Scaling 6 arms	**1.6280** %	1.9251 %	68.8295 %

For the second setting, due to the size of the state space, a complete analysis of the performance of the algorithms throughout their training is not feasible. Therefore, we decided to evaluate the performance of the algorithms in the last iteration, where they are expected to have converged to an optimal policy. The data presented in Table 5.1 shows the relative performance of LPCA-DE, LPCA-Greedy, and DDPG with OptLayer algorithms.

For the 6-arm speed scaling scenario, LPCA DE has a relative error of 1.6280%, while LPCA Greedy has a slightly higher error of 1.9251%. In contrast, DDPG with OptLayer has a significantly higher relative error of 18.8295%. This significant difference, present in all environments and settings, shows the superior efficiency of the LPCA algorithms in closely approximating the theoretical performance, even in a complex and resource-intensive environment.

Chapter 6

Conclusion and Future Directions

6.1 Summary of Key Findings

In the field of discrete actions within Restless Multi-Armed Bandit (RMAB) problems, the development and implementation of the QWI and QWINN algorithms stand out as significant achievements of this thesis. These two-timescale algorithms are designed to compute the Whittle indices accurately, converging closely to the exact theoretical values. The convergence rate of QWINN is shown to be superior to that of QWI, due to the integration of neural networks that improve the convergence of Q-values across all states. This fast and accurate index computation is particularly evident in smaller scale environments, providing a robust foundation for the algorithms' effectiveness.

Throughout various test scenarios, both QWI and QWINN demonstrated consistent performance in terms of index convergence, as illustrated by Spearman's rank correlation metrics. This consistency was observed regardless of the number of arms in the problem, demonstrating a clear performance advantage over traditional reinforcement learning methods such as DQN and tabular Q-learning as shown in Section 4.3.3.1. In environments characterized by large state spaces, these algorithms not only matched, but occasionally surpassed the performance of the previous state-of-the-art, NeurWIN, particularly in reducing absolute error, as highlighted in Figure 4.6 left.

The practical impact of these algorithms is further illustrated by their performance in larger environments, such as the extended restart and circular models of section 4.3.3.2. In these environments, the rapid alignment of computed indices with theoretical Whittle indices allowed for faster convergence to the optimal policy performance. In particular, in the extended circular problem with state spaces as large as $|\mathcal{S}_i| = \{10, 50\}$, NeurWIN struggled with correct index ordering-a challenge handled successfully by QWINN and even QWI, which effectively captured the appropriate index order critical for policy optimization, as shown in Figures 4.9 left and right.

Furthermore, in the extended restart problem, QWINN consistently identified the correct ordering of key states, a capability shown in Figure 4.10 top right and bottom right. Even QWI, which uses a tabular approach, demonstrated superior long-term performance over NeurWIN by achieving better index convergence in essential states, as shown in Figures 4.12 left and right.

A notable advantage of QWI, and especially QWINN, is their accuracy in estimating Whittle indices in hybrid RMAB environments where arms may have different transition probabilities and reward functions, as explored in Section 4.3.3.3. This is evident as QWINN maintains close index orders across all states, in sharp contrast to NeurWIN's performance, which is competitive in homogeneous environments, as shown in Figure 4.7 top left, but struggles in heterogeneous environments, as shown in Figure 4.13 bottom. Here, QWINN's performance was significantly closer to the theoretical Whittle Index threshold, emphasizing its robust adaptability and accuracy in more complex scenarios.

Together, these results highlight the significant advances that QWI and QWINN have brought to the computation of Whittle indices for RMAB problems.

In our continuous action analysis, we have extended the algorithms of Niño-Mora [45] and Weber [46] for numerically computing Whittle indices to handle multi-actions with any kind of discretization in the action space in Section 5.1. This advancement allows us to compute Whittle indices for continuous action problems with varying levels of discretization. By integrating this with the policy heuristic introduced in Section 5.1.1, we can approximate a fully continuous action Whittle index policy.

A key part of our research is the development of the Lagrangian Policy for Continuous Action (LPCA) algorithm, an advanced approach for solving continuous action reinforcement learning in weakly coupled problems, introduced in Section 5.2. LPCA utilizes a Lagrangian relaxation to effectively decouple these problems, enabling the optimization of expected rewards within given resource constraints. The algorithm employs two distinct strategies: a differential evolution algorithm designed to navigate the action space within these constraints, and a greedy algorithm that sequentially allocates resources to optimize performance rewards.

Our experimental results in Section 5.3.3 demonstrate that both versions of LPCA consistently outperform DDPG with OptLayer in all tested environments and configurations, achieving policy performances that closely match the theoretical Whittle indices derived from our numerical evaluations. LPCA exhibits remarkable resource allocation efficiency under conditions of resource scarcity, highlighting its utility in critical scenarios.

Moreover, LPCA effectively handles both simple and complex problem structures. In simpler scenarios, such as the Type A problem, LPCA directs resources to specific arms based on their state. In more complex environments, such as the Admission Control and Speed Scaling problems, LPCA balances the distribution of resources across states to maintain equilibrium between the MDPs, significantly improving overall system performance. This nuanced resource management contrasts sharply with DDPG with OptLayer, which is competent in simpler settings such as Type A, but generally falters in more complex configurations.

In conclusion, the introduction of LPCA not only advances the computational methodology for handling continuous action spaces in RMAB problems but also sets a new benchmark for integrating resource constraints into policy computation. This establishes a robust framework for future research and applications in this area, demonstrating the potential for LPCA to address complex decision-making problems under uncertainty and resource limitations.

6.2 Future Directions and Potential Developments

Several promising avenues for improving our algorithms emerge as we consider the future directions and potential developments stemming from this thesis. These ideas, with a particular focus on the QWINN and LPCA algorithms, aim to improve the robustness and applicability of the models we developed.

For QWINN, a notable improvement would be the implementation of a dual neural network system - one dedicated to computing Q-values and another specialized for Whittle index computation. The existing hybrid model, while efficient and computationally light, could never handle continuous action spaces. By allowing the action to be an input to the Whittle index neural network, the system could calculate indices, while taking advantage of the stability of the two-time scale method to improve accuracy and robustness. This development could be a significant step towards a continuous action variant of QWINN, enhancing its utility in more dynamic environments. Although the first steps towards this innovation have been taken, considerable work remains to be done, mainly due to time constraints on the initial research.

Turning to the LPCA algorithm, the recoupling of the state space to compute the action policy dictionary is a significant computational challenge. This process is critical as it integrates the decoupled Q-values back into a unified policy, but it becomes increasingly expensive as the number of MDPs grows. Future research could explore ways to optimize or completely redesign this recoupling process to reduce the overhead.

In addition, we initiated research on the computation of the analytical continuous Whittle index, which revealed intriguing possibilities for improving index computation without relying on numerical discretization methods. This approach promises improved precision and accuracy in dynamic environments with continuous action spaces. However, the indexability properties of the algorithm require further investigation to fully realize its potential.

Appendix A

Appendix

A.1 Theoretical Foundations of QWI

Consider the following definitions:

Definition A.1. *A function is considered Lipschitz if it satisfies the condition that for any two points x, y in its domain, there exists a constant $L > 0$ such that $|f(x) - f(y)| \leq L|x - y|$. This property ensures the function's output changes at a controlled rate in response to variations in the input, guaranteeing continuity and, if differentiable, a bounded derivative.*

Definition A.2. *A σ-field is a mathematical structure that encompasses a collection of sets, including the entire sample space, effectively organizing the conceivable events up to a given time point*

Definition A.3. *An increasing σ-field, $\mathcal{F}_n$, symbolizes a sequence of σ-fields where each subsequent field incorporates all prior information, mirroring the accumulation of knowledge over time.*

Definition A.4. *A martingale difference sequence with respect to increasing σ-fields is a sequence M_n that meets the condition $\mathbb{E}[M_{n+1}|F_n] = 0$ almost surely for all n. This property indicates that, given the information up to the present, the expected change at the next step is zero, implying that future updates do not systematically favor any direction.*

Given these descriptions, we shall abstract our notation of Q-values and Whittle indices to develop the theoretical foundation for this convergence. Let us consider the following iterations

$$x_{n+1} = x_n + \alpha(n)(h(x_n, y_n) + M^{(1)}_{n+1}) \tag{A.1}$$

$$y_{n+1} = y_n + \beta(n)(g(x_n, y_n) + M^{(2)}_{n+1}) \tag{A.2}$$

where h and g are Lipschitz functions and $M_n^{(1)}, M_n^{(2)}$ are martingale differences. The step sizes $\alpha(n), \beta(n)$ are chosen to satisfy specific conditions ensuring

$$\sum_n \alpha(n) = \sum_n \beta(n) = \infty, \sum_n (\alpha(n)^2 + \beta(n)^2) < \infty, \frac{\beta(n)}{\alpha(n)} \to 0,$$

that is, $\beta(n)$ diminishes to zero faster than $\alpha(n)$, reflecting the slower update timescale of one of the stochastic processes.

This differential pacing is analogous to a system of O.D.E.s, where the fast time-scale updates represent the transient behavior, while the slow time-scale updates represent the long-term dynamics:

$$\dot{x}(t) = \frac{1}{\epsilon} h(x(t), y(t)) \tag{A.3}$$

$$\dot{y}(t) = g(x(t), y(t)) \tag{A.4}$$

in the limit $\epsilon \downarrow 0$. Under this analogy, the system exhibits behavior where $x(t)$ rapidly adjusts to changes, while y is fixed.

The theoretical model requires several assumptions to ensure the convergence of (x_n, y_n) to a stable equilibrium.

A (A.3) has a globally asymptotically stable equilibrium $\lambda(y)$ where $\lambda : \mathbb{R}^k \to \mathbb{R}^d$ is a Lipschitz map. For small values of ϵ we can expect $x(t)$ to track $\lambda(t)$. $\dot{y}(t) = g(\lambda(y(t)), y(t))$ captures the behavior of $y(\cdot)$ in (A.4).

B (A.4) has a globally asymptotically stable equilibrium y^*.

Equations (A.3) and (A.4) define an ODE system where the solution trajectory $(x(t), y(t))$ is expected to converge to the equilibrium point $(\lambda(y^*), y^*)$. This convergence property can be extended to Equations (A.2) and (A.1), where Equation (A.2) represents a quasi-static view of Equation (A.1), and vice versa, Equation (A.1) can be interpreted as an almost-averaged representation of Equation (A.2).

C $\sup_n(\|x_n\| + \|y_n\|) < \infty$, where $\|\cdot\|$ denotes the Euclidean norm. This condition ensures that the sequence (x_n, y_n) remains bounded, preventing the system from diverging.

Lemma A.1. $(x_n, y_n) \to \{(\lambda(y), y) : y \in \mathbb{R}^d\}$.

Proof. Consider the original update equation (A.2) for y_n reformulated as:

$$y_{n+1} = y_n + \alpha(n)(\epsilon_n + M_{n+1}^{(3)}) \tag{A.5}$$

where $\epsilon_n = \frac{\beta(n)}{\alpha(n)} g(x_n, y_n)$ represents the rescaled Lipschitz function and $M_{n+1}^{(3)} = \frac{\beta(n)}{\alpha(n)} M_{n+1}^{(2)}$ denotes a rescaled martingale difference sequence, indexed by $n \geq 0$.

This reformulation aligns with the two-timescale approach by adjusting the update equation for y_n to explicitly reflect the slower update rate through the term ϵ_n, which diminishes as n increases. The interaction between the update equations for x_n and y_n can be described by the pair of o.d.e: $\dot{x}(t) = h(x(t), y(t)), \dot{y}(t) = 0$, where the dynamics of $x(t)$ are influenced by the current state of $y(t)$, but $y(t)$ itself remains constant over the infinitesimal time increments considered in the o.d.e. framework. As a result, $\|x_n - \lambda(y_n)\| \to 0$ as $n \to \infty$, as $\{x_n\}$ tracks the trajectory of $\{\lambda(y_n)\}$.

Theorem A.1. $(x_n, y_n) \to (\lambda(y^*), y^*)$

Proof. To demonstrate this convergence, consider a sequence $s(n)$ initialized at $s(0) = 0$ and defined for $n \geq 1$ by $s(n) = \sum_{i=0}^{n-1} \beta(i)$. Additionally, define a zero mean square-integrable martingale sequence $\psi_n = \sum_{m=0}^{n-1} \frac{\beta(n)}{\alpha(n)} M_{m+1}^{(2)}$, for $n \geq 1$. A piecewise linear continuous function $\tilde{y}(t)$, where $\tilde{y}(s(n)) = y_n$ for $t \geq 0$, linearly interpolates the values of y_n across intervals $[s(n), s(n+1)]$. With $[t]'$ denoting $\max s(n) : s(n) \leq t$ for $t \geq 0$, we can express $\tilde{y}(s(n+m))$ as a sum of integrals and sums capturing the dynamic and stochastic components of the updates.

$$\begin{aligned}\tilde{y}(s(n+m)) = \tilde{y}(s(n)) &+ \int_{s(n)}^{s(n+m)} g(\lambda(\tilde{y}(t)), \tilde{y}(t))dt \\ &+ \int_{s(n)}^{s(n+m)} (g(\lambda(\tilde{y}([t]')), \tilde{y}([t]')) - g(\lambda(\tilde{y}(t)), \tilde{y}(t)))dt \qquad \text{(a)} \\ &+ \sum_{k=1}^{m-1} b(n+k)(g(x_{n+k}, y_{n+k}) - g(\lambda(y_{n+k}), y_{n+k})) \qquad \text{(b)} \\ &+ (\psi_{n+m+1} - \psi_n). \qquad \text{(c)}\end{aligned} \tag{A.6}$$

The comparison of $\tilde{y}(t)$ with the trajectory $y^s(t)$ of the differential equation $\dot{y}(t) = g(\lambda(y(t)), y(t))$, starting from $y^s(s) = \tilde{y}(s)$, yields an upper bound on their discrepancy over any interval $[s, s+T]$ through the application of the Gronwall Inequality [73].

$$\sup_{[t \in s, s+T]} \|\tilde{y}(t) - y^s(t)\| \leq KT((a) + (b) + (c)) \tag{A.7}$$

This upper bound encompasses discretization error (a), tracking error (b), and noise error (c). Each of those terms evolve as $O(\sum_{k \geq n} b(k)^2)$, $O(\sup_{k \geq n} \|x_k - \lambda(y_k)\|)$ and $O(\sup_{k \geq n} \|\psi_k - \psi_n\|)$ respectively, and all of them diminish to zero as $s \to \infty$, ensuring the convergence of $\tilde{y}(t)$ to $y^s(t)$ in the supremum norm over any finite interval.

$$\sup_{t \in [s, s+T]} \|\tilde{y}(t) - y^s(t)\| \to 0$$

Following Borkar's [60] Theorem 2 from Chapter 2, this analysis leads to the conclusion that y_n converges to y^*, and by Lemma A.1, x_n converges to $\lambda(y^*)$, thereby establishing the theorem's claim.

An alternative approach to implementing this two-timescale method involves using the same learning rate $\alpha(n)$ for both (A.1) and (A.2) updates but selectively applying the (A.2) updates only at a subsequence of iterations $n(k)$, where the interval between successive updates grows unbounded, i.e. $n(k+1) - n(k) \to \infty$ as $k \to \infty$.

In practice, a hybrid strategy that uses both techniques offers the most stable and effective policy for simultaneous updates. We can achieve this by choosing step sizes $\alpha(n)$ and $\beta(n)$ such that $\beta(n)$ corresponds to a slower timescale and applying updates to (A.2) in a subsequence $nN, n \geq 0$ for an integer $N > 1$, while keeping y_n constant between these updates.

A.2 Proof of convergence of QWI algorithm

Equations (4.6) and (4.7) naturally lead to the abstract two-timescale stochastic approximation method in Equations (A.2) and (A.1). We define $F_{su}^{\lambda}(\Psi(j,b))$ and $M_{n+1}(s,u)$ as follows:

$$\begin{aligned} F_{sa}^{\lambda}(\Psi(j,b)) = (1-a)(R_0(s)+\lambda) + aR_1(s) + \\ \gamma \sum_j p(j|i,a) \max_{v\in\{0,1\}} \Psi(j,v) \end{aligned} \tag{A.8}$$

and

$$\begin{aligned} M_{n+1}(s,a) = (1-a)(R_0(s)+\lambda_n(x)) + aR_1(s) + \\ \max_{v\in\{0,1\}} Q_n(x_{n+1},v) - F_{sa}^{\lambda_n(x)}(Q_n) \end{aligned} \tag{A.9}$$

With these definitions, the Q-value update in (4.6) can be expressed as:

$$Q_{n+1}^x(s,a) = Q_n^x(s,a) + \alpha(n)\left[F_{sa}^{\lambda_n(x)}(Q_n) - Q_n + M_{n+1}(s,a)\right] \tag{A.10}$$

Comparing equations (4.6) and (A.10), we obtain:

- $a(n) = \alpha(n)$
- $h(x_n,y_n) = F_{sa}^{\lambda_n(x)}(Q_n) - Q_n$, where $x_n = Q_n$ and $y_n = \lambda_n$ are the Q-value and Whittle index estimate respectively
- $M_{n+1}(s,a)$ is the martingale difference sequence $M_{n+1}^{(1)}$

Similarly, the equivalence of equations (4.7) and (A.2) leads to:

- $b(n) = \beta(n)$
- $g(x_n,y_n) = Q_n^x(x,1) - Q_n^x(x,0)$
- The martingale difference sequence $M_{n+1}^{(2)} = 0$

Assuming the boundedness of Equations (4.6) and (4.7), a condition we will establish subsequently, we commence by rephrasing the equation for computing the Whittle indices (4.7) as follows:

$$\lambda_{n+1}(x) = \lambda_n(x) + \alpha(n)\left(\frac{\beta(n)}{\alpha(n)}\right)(Q_n^x(x,1) - Q_n^x(x,0)), \tag{A.11}$$

To capture the evolution of Q_n^x and $\lambda_n(x)$, we introduce $\bar{Q}(t)$ and $\bar{\lambda}(t)$ as piecewise linear continuous functions interpolating their trajectories over each interval $[\tau(n),\tau(n+1)]$, for $n \geq 0$, where $\tau(n) = \sum_{m=0}^{n} \alpha(m)$ for $m \geq 0$.:

$$\bar{Q}(t) = Q(n) + \left(\frac{t - \tau(n)}{\tau(n+1) - \tau(n)}\right)(Q(n+1) - Q(n)) \tag{A.12}$$

$$\bar{\lambda}(t) = \lambda(n) + \left(\frac{t - \tau(n)}{\tau(n+1) - \tau(n)}\right)(\lambda(n+1) + \lambda(n)) \tag{A.13}$$

for $t \in [\tau(n), \tau(n+1)]$.

These functions reflect the asymptotic dynamics of the system described by

$$\dot{Q}(t) = h(Q(t), \lambda(t)), \dot{\lambda} = 0$$

with the second equation stemming from the fact that $\frac{\beta(n)}{\alpha(n)} \to 0$ as per (A.11). From the perspective of $Q(t)$, $\lambda(\cdot)$ effectively acts as a constant, leading to the simplified differential equation $\dot{Q} = h(Q(t), \lambda')$. This system is both well-defined and bounded, converging to a stable equilibrium Q^*_λ, as outlined in Theorem 3.4, p. 689 in [74], ensuring that $Q^x_n - Q^*_{\lambda_n} \to 0$ as $n \to \infty$.

For the trajectory of $\lambda(t)$, we define an alternative timescale via:

$$\tilde{\lambda}(t) = \lambda(n) + \left(\frac{t - \tau'(n)}{\tau'(n+1) - \tau'(n)}\right)(g(n+1) - g(n)), \tag{A.14}$$

for $t \in [\tau'(n), \tau'(n+1)]$, where $\tau'(n) = \sum_{m=0}^{n} \beta(m)$ for $n \geq 0$. This alternative trajectory aligns with the differential equation:

$$\dot{\Lambda}(t) = Q^*_{\Lambda(t)}(x, 1) - Q^*_{\Lambda(t)}(x, 0)$$

In this framework, when $\Lambda(t) > \lambda(x)$ (indicating an excess subsidy), the passive mode becomes preferable, driving $\Lambda(t)$ downwards. Conversely, should $\Lambda(t) < \lambda(x)$, $\Lambda(t)$ ascends. Consequently, $\Lambda(\cdot)$'s trajectory remains within bounds. Given its well-defined and bounded nature, this scalar differential equation converges to a stable equilibrium where $Q^{(}_{\Lambda}x, 1) = Q^{(}_{\Lambda}x, 0)$, identifying the equilibrium point as the Whittle index, where both action policies are equally favorable.

Bibliography

[1] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966. See page 2.

[2] Richard Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957. See pages 2, 4.

[3] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989. See page 3.

[4] Edward L. Thorndike. The Law of Effect. *The American Journal of Psychology*, 39(1/4):212–222, 1927. See page 3.

[5] B. F. Skinner. *The Behavior of Organisms: An Experimental Analysis*. B. F. Skinner Foundation, December 2019. Google-Books-ID: S9WNCwAAQBAJ. See page 3.

[6] Alan M Turing. Intelligent machinery, a heretical theory. *Philosophia Mathematica*, 4(3):256–260, 1996. See page 3.

[7] Marvin Minsky. Steps toward Artificial Intelligence. *Proceedings of the IRE*, 49(1):8–30, January 1961. See page 3.

[8] Michael Lvovitch Tsetlin. On behaviour of finite automata in random medium. *Avtomat. i Telemekh*, 22(10):1345–1354, 1961. See page 3.

[9] A. Harry Klopf. The hedonistic neuron : a theory of memory, learning, and intelligence. *(No Title)*. See page 3.

[10] A. Harry Klopf. *Brain Function and Adaptive Systems: A Heterostatic Theory*. Air Force Cambridge Research Laboratories, Air Force Systems Command, United States Air Force, 1972. Google-Books-ID: QExVCsabOasC. See page 3.

[11] A. Harry Klopf. A comparison of natural and artificial intelligence. *ACM SIGART Bulletin*, (52):11–13, June 1975. See page 3.

[12] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959. See page 3.

[13] A. Harry Klopf. A neuronal model of classical conditioning. *Psychobiology*, 16(2):85–125, June 1988. See page 3.

[14] Richard S. Sutton and Andrew G. Barto. Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review*, 88(2):135–170, 1981. See page 3.

[15] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, September 1983. See page 3.

[16] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, August 1988. See page 3.

[17] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992. See pages 4, 14, and 32.

[18] Paul J. Werbos. Building and Understanding Adaptive Systems: A Statistical/Numerical Approach to Factory Automation and Brain Research. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(1):7–20, January 1987. See page 4.

[19] Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995. See page 4.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. Technical report, December 2013. arXiv:1312.5602 [cs] type: article. See pages 4, 38.

[21] Abubakar Sadiq Abdulhameed and Serhii Lupenko. Potentials of reinforcement learning in contemporary scenarios. *Bulletin of the Ternopil National Technical University*, 106(2):92–100, 2022. See page 4.

[22] Maroning Useng and Suleiman Abdulrahman. A Survey on Distributed Reinforcement Learning. *Mesopotamian Journal of Big Data*, 2022:44–50, November 2022. See page 4.

[23] Ali Alameer, Haitham Saleh, and Khaled Alshehri. Reinforcement learning in quantitative trading: A survey. *Authorea Preprints*, 2023. See page 4.

[24] Ryan Lowe, YI WU, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. See page 4.

[25] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):178:7234–178:7284, January 2020. See page 4.

[26] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning. Technical report, June 2017. arXiv:1706.05296 [cs] type: article. See page 4.

[27] Andrei Andreevich Markov. Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete*, 15(135-156):18, 1906. See page 4.

[28] Francisco Robledo, Vivek Borkar, Urtzi Ayesta, and Konstantin Avrachenkov. QWI: Q-learning with Whittle Index. *ACM SIGMETRICS Performance Evaluation Review*, 49(2):47–50, January 2022. See page 7.

[29] Francisco Robledo Relaño, Vivek Borkar, Urtzi Ayesta, and Konstantin Avrachenkov. Tabular and deep learning for the whittle index. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2024. See pages 7, 43, and 46.

[30] Francisco Robledo, Urtzi Ayesta, and Konstantin Avrachenkov. Deep reinforcement learning for weakly coupled MDP's with continuous actions. June 2024. See pages 7, 77.

[31] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. See page 9.

[32] Ronald A. Howard. *Dynamic programming and Markov processes*. Dynamic programming and Markov processes. John Wiley, Oxford, England, 1960. See page 11.

[33] David Blackwell. Discrete Dynamic Programming. *The Annals of Mathematical Statistics*, 33(2):719–726, 1962. See page 11.

[34] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954. See page 12.

[35] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994. See pages 14, 33.

[36] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933. See pages 14, 15.

[37] Herbert Robbins. Some aspects of the sequential design of experiments. 1952. See pages 14, 15.

[38] Richard Bellman. A Problem in the Sequential Design of Experiments. *Sankhyā: The Indian Journal of Statistics (1933-1960)*, 16(3/4):221–229, 1956. See page 15.

[39] John C Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society: Series B (Methodological)*, 41(2):148–164, 1979. See pages 15, 17, 20, and 42.

[40] Manjari Asawa and Demosthenis Teneketzis. Multi-armed bandits with switching penalties. *IEEE Transactions on Automatic Control*, 41(3):328–348, March 1996. See page 15.

[41] Peter Whittle. Multi-Armed Bandits and the Gittins Index. *Journal of the Royal Statistical Society: Series B (Methodological)*, 42(2):143–149, 1980. See page 18.

[42] Peter Whittle. Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability*, 25(A):287–298, January 1988. See pages 18, 20, and 41.

[43] Jeffrey Thomas Hawkins. *A Langrangian decomposition approach to weakly coupled dynamic optimization problems and its applications*. PhD thesis, Massachusetts Institute of Technology, 2003. See pages 24, 25, 27, 77, 78, and 79.

[44] Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997. See page 24.

[45] José Niño-Mora. Dynamic priority allocation via restless bandit marginal productivity indices. *TOP*, 15(2):161–198, December 2007. See pages 28, 29, 74, 75, and 96.

[46] Richard Weber. Comments on: Dynamic priority allocation via restless bandit marginal productivity indices. *Top*, 15(2):211–216, 2007. See pages 29, 74, 75, and 96.

[47] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943. See page 34.

[48] Paul Werbos and Paul John. Beyond regression : new tools for prediction and analysis in the behavioral sciences /. January 1974. See page 36.

[49] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. Technical report, January 2017. arXiv:1412.6980 [cs] type: article. See pages 37, 46.

[50] Shunichi Amari. A Theory of Adaptive Pattern Classifiers. *IEEE Transactions on Electronic Computers*, EC-16(3):299–307, June 1967. See page 38.

[51] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. See page 38.

[52] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning internal representations by error propagation, 1985. See page 38.

[53] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. See page 38.

[54] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015. See page 38.

[55] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. See page 38.

[56] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. Technical report, February 2016. arXiv:1511.05952 [cs] type: article. See page 39.

[57] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling Network Architectures for Deep Reinforcement Learning. pages 1995–2003. PMLR, June 2016. See page 39.

[58] Jing Fu, Yoni Nazarathy, Sarat Moka, and Peter G Taylor. Towards q-learning the whittle index for restless bandits. In *2019 Australian & New Zealand Control Conference (ANZCC)*, pages 249–254. IEEE, 2019. See pages 42, 51.

[59] Khaled Nakhleh, Santosh Ganji, Ping-Chun Hsieh, I-Hong Hou, and Srinivas Shakkottai. NeurWIN: Neural Whittle Index Network For Restless Bandits Via Deep RL. In *Advances in Neural Information Processing Systems*, volume 34, pages 828–839. Curran Associates, Inc., 2021. See pages 42, 50.

[60] V. S. Borkar. Stochastic approximation: a dynamical systems viewpoint. *Springer*, 48, 2009. See pages 43, 102.

[61] Konstantin E. Avrachenkov and Vivek Shripad Borkar. Whittle index based Q-learning for restless bandits with average reward. *Automatica*, 139:110186, May 2022. See page 51.

[62] Zhe Yu, Yunjian Xu, and Lang Tong. Deadline scheduling as restless bandits. *IEEE Transactions on Automatic Control*, 63(8):2343–2358, 2018. See pages 51, 53, and 54.

[63] José Niño-Mora. Markovian Restless Bandits and Index Policies: A Review. *Mathematics*, 11(7):1639, January 2023. See page 54.

[64] Christopher J. Conselice, Aaron Wilkinson, Kenneth Duncan, and Alice Mortlock. The evolution of galaxy number density at z < 8 and its implications. *The Astrophysical Journal*, 830(2):83, October 2016. See page 54.

[65] *Spearman Rank Correlation Coefficient*, pages 502–505. Springer New York, New York, NY, 2008. See page 56.

[66] Donald E. Kirk. *Optimal Control Theory: An Introduction*. Courier Corporation, January 2004. Google-Books-ID: fCh2SAtWIdwC. See page 78.

[67] Dimitri Bertsekas. *Dynamic Programming and Optimal Control: Volume I*. Athena Scientific, 2012. Google-Books-ID: qVBEEAAAQBAJ. See page 78.

[68] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016. See pages 80, 85.

[69] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, February 2011. See page 82.

[70] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. Technical report, July 2019. arXiv:1509.02971 [cs, stat] type: article. See page 85.

[71] Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. OptLayer - Practical Constrained Optimization for Deep Reinforcement Learning in the Real World. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6236–6243, May 2018. ISSN: 2577-087X. See page 86.

[72] Jackson A. Killian, Arpita Biswas, Sanket Shah, and Milind Tambe. Q-Learning Lagrange Policies for Multi-Action Restless Bandits. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pages 871–881, New York, NY, USA, August 2021. Association for Computing Machinery. See page 86.

[73] T. H. Gronwall. Note on the Derivatives with Respect to a Parameter of the Solutions of a System of Differential Equations. *Annals of Mathematics*, 20(4):292–296, 1919. See page 101.

[74] Jinane Abounadi, Dimitrib Bertsekas, and Vivek S Borkar. Learning algorithms for markov decision processes with average cost. *SIAM Journal on Control and Optimization*, 40(3):681–698, 2001. See page 104.