Constraint- and Heuristic-Aware Deep Reinforcement Learning for Infrastructure Intervention Planning

Master thesis submitted to Delft University of Technology in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

Management of Technology
Faculty of Technology, Policy and Management
Delft University of Technology

Student name: **Ángel Villar Montero**
Student number: **6077447**

Graduation committee

Omar Kammouh, SE & S	First supervisor
Aaro Ding, ICT	Chair & Second supervisor
Charalampos Andriotis	Advisor
Christos Lathourakis, TNO	External advisor
Andrés Martínez Colán, TNO	External advisor

TU Delft Delft University of Technology

"Logic is the anatomy of thought."
John Locke

Abstract

As urban infrastructure continues to age, municipalities face increasingly complex challenges in maintaining critical assets under financial, safety, and societal constraints. In Amsterdam, more than 200 km of historic quay walls, many of them several centuries old, require carefully coordinated interventions to prevent structural failure while limiting urban disruption and long-term costs. Traditional maintenance planning approaches are typically reactive or rule-based, and struggle to account for stochastic degradation processes and the non-linear interaction costs associated with simultaneous street and traffic closures in dense urban networks.

This thesis proposes a multi-agent Deep Reinforcement Learning (DRL) framework for generating optimal, constraint-aware maintenance policies. The framework combines DRL with structural risk assessments and network-level societal impacts quantified through the Urban Strategy digital twin. To reflect the operational realities of public asset management, multiple constraint-handling mechanisms are integrated into the learning process, ranging from hard, component-level constraints to soft, network-level constraints, jointly capturing financial, safety, and societal considerations.

Experimental results show that the DRL agent consistently converges to optimal or near-optimal policies while satisfying both hard and soft operational constraints, and significantly outperforms existing reactive maintenance strategies. Benchmarking against classical optimization methods demonstrates that the DRL approach maintains near-optimal performance as the problem scales to higher-dimensional environments where traditional methods become computationally intractable. Furthermore, warm-start initialization via curriculum learning is shown to substantially improve training stability, convergence speed, and solution quality in constrained settings.

Overall, this research demonstrates that constraint-aware DRL can serve as a powerful and flexible decision-support tool for urban infrastructure management. By bridging the gap between theoretical optimization and real-world operational constraints, the proposed framework provides a scalable foundation for future applications in city-scale asset management and related domains.

Acknowledgments

Many months ago, while searching for a thesis project, I came across an opportunity involving Deep Reinforcement Learning on the graduation portal. Despite my very limited experience in the topic, I chose to embrace the challenge rather than follow a safer path more aligned with my previous background. Looking back, I am truly glad I made that decision. This journey has allowed me to grow both academically and personally, and I am deeply grateful to the many people who supported me along the way.

First, I would like to sincerely thank my mentors, Omar Kammouh, for his continuous guidance and support, and Charalampos Andriotis, for sharing his knowledge and expertise during the project. I am also thankful to Aaron Ding for his helpful feedback, which improved several aspects of the work.

I would like to extend my special thanks to my supervisors at TNO, Andrés Martínez Colán and Christos Lathourakis. Your support was invaluable, not only in tackling the technical challenges of the project, but also in the many conversations, reflections, and moments we shared. Beyond being exceptional professionals, you are truly great people, and working with you has been a pleasure.

Last but certainly not least, I would like to thank my friends and family — those in Spain, in the Netherlands, and scattered across the world — who have been a constant source of strength and joy not only throughout this journey, but in every adventure of my life.

To my friends from Galicia, who will always be a part of who I am. To Puchol, Ramón, Héctor, Yorch, Mateo, Pablito, and Gonzalo, for making me feel like at home in Delft from the very beginning. To Julia, Pascal, Nuria, and Elisa, for turning a house into a home, for the yoga morning sessions, and for making everyday life brighter and lighter. And to Crina, for her patience, her constant support, and for being such an essential presence in my life throughout this past year.

Finally, to my mother, my father, and my aunt: who have been the foundation of everything I have achieved. Thank you for your unconditional love and support in every decision I make, for always being there for me, and for believing in me even when I doubt myself. Any success in my life carries a part of you. And to my sister Rosa, thank you not only for all of the above, but also for understanding me like no one else, for always standing by me, and for being a constant source of inspiration and guidance in my life.

To all of you who have walked this path with me in one way or another—this achievement is as much yours as it is mine. Thank you.

Ángel Villar Montero
Delft, February 2026

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ARK Amsterdamse Risicobeoordeling Kademuren.

CEM Cross-Entropy Method.

DCMAC Deep Centralized Multi-Agent Actor Critic.

DDMAC Deep Decentralized Multi-Agent Actor-Critic.

DRL Deep Reinforcement Learning.

GAE Generalized Advantage Estimation.

KL Kullback-Leibler.

MARL Multi-Agent Reinforcement Learning.

MDPs Markov Decision Processes.

MOT Management of Technology.

PID proportional–integral–derivative.

POMDPs Partially Observable Markov Decision Processes.

PPO Proximal Policy Optimization.

RCPO Reward Constrained Policy Optimization.

RL Reinforcement Learning.

TNO Netherlands Organisation for Applied Scientific Research.

TRPO Trust Region Policy Optimization.

US Urban Strategy.

VMA Verkeersmodel Amsterdam.

1 Introduction

1.1 Motivation

In the complex world we live in, managing infrastructure systems is a critical and increasingly challenging task. As aging infrastructure across the globe approaches or exceeds its intended service life, the need for effective maintenance strategies becomes more important than ever. However, this is a very challenging problem. Components wear out in unpredictable ways. Systems are complex and interconnected. Inspections are uncertain. This is even more difficult in cities, where things get even more complicated — roads, bridges, and utilities must be maintained together, often with limited resources.

A good example is the quay wall network in Amsterdam. While the city is responsible for around 600 km of quay walls in total, more than 200 km of these form the historic city center, with many structures being over 100 years old, and some as old as 300 [8]. They support streets, homes, shops, and public spaces. They're part of the city's identity and daily life. If they fail, the impact can be huge — not just in repair costs, but also in traffic disruption, safety risks, and effects on local businesses and residents.

Even with their importance, like for most of the infrastructure around the world, the way maintenance is planned is still quite basic. Quay wall maintenance is mostly done using simple, rule-based or reactive methods. These include time-based schedules, condition checks, or thresholds that trigger repairs [9]. While straightforward, these strategies don't account for long-term effects. This can lead to poor timing, either waiting too long (which raises risks and costs) or acting too early (which wastes resources). They also miss the chance to coordinate tasks or take advantage of cost-saving synergies between components.

To correct this, optimization techniques have been proposed. Mathematical programming — like nonlinear models, mixed-integer programming, and heuristic algorithms — can help plan and group maintenance actions to lower costs and reduce disruption [10] [11]. But these methods don't scale well. As the number of components and actions grow, the computations get too heavy. This makes them hard to use for large systems like a whole city. On top of that, many models assume full knowledge of how things degrade or ignore uncertainty, which limits how useful they are.

Sequential decision-making frameworks offer an alternative way to model infrastructure maintenance problems by explicitly accounting for uncertainty, long-term consequences, and trade-offs between competing objectives. In recent years, Reinforcement Learning (RL) has emerged as a promising data-driven approach for such problems, as it allows policies to be learned through interaction with a simulated environment rather than relying on fully specified analytical models [12]. However, classical RL methods struggle when faced with large state and action spaces, slow convergence, and limited generalization. Issues that are unavoidable in realistic infrastructure systems.

Deep Reinforcement Learning (DRL) addresses these challenges by combining RL with neural network function approximation, enabling learning in high-dimensional and complex environments [13]. Recent studies have demonstrated the potential of DRL for infrastructure maintenance planning [14, 15], including inspection scheduling, intervention optimization, and life-cycle cost reduction. Nevertheless, most existing applications remain exploratory and are often tested on simplified or idealized settings.

Applying DRL techniques to infrastructure planning is a relatively new but fast-growing research direction. Key publications in this field are recent, highlighting how active and emergent this area is. That's why this project wants to build on this cutting-edge line of work, focusing on a case study of the inner-city quay walls in Amsterdam. The quay walls were chosen for their importance to the city, the availability of rich data, and because a simulation implementation using Urban Strategy (US) [16] was already available.

This thesis was carried out in collaboration with Netherlands Organisation for Applied Scientific Research (TNO) and builds on their research into data-driven maintenance planning for urban infrastructure. TNO defined the problem setting and provided an initial simulation environment along with a DRL-based model for quay wall maintenance in Amsterdam. While this model demonstrated the potential of DRL for supporting maintenance planning, it required further development and refinement before it could serve as a reliable foundation for the work presented in this thesis.

The initial model also remained relatively simplistic. A major limitation was its lack of incorporation of real-world constraints and heuristics that asset managers typically consider, such as annual budgets, spatial coordination, or thresholds for risk and operational hindrances. These elements are central to realistic infrastructure management, and their absence limits the model's suitability for practical decision-making.

This thesis therefore focused on enhancing both the robustness and realism of the existing DRL framework by integrating explicit constraints and domain-relevant heuristics into the decision-making process. Addressing constraints in this context is non-trivial. While some academic work applies simple reward penalties (known as reward shaping), this approach is often unstable, difficult to tune, and does not reliably enforce constraints. In contrast, this research implements more principled methods, such as Lagrangian relaxation and action masking, selecting the most appropriate approach for each constraint.

Incorporating these constraint-handling techniques required more than merely adjusting the reward function; it necessitated substantial modifications to both the agent architecture and the training procedure. Given that multi-agent actor-critic frameworks—specifically the Deep Centralized Multi-Agent Actor Critic (DCMAC) method—have previously demonstrated strong capabilities in managing the complex, distributed dynamics of unconstrained infrastructure systems, this thesis deliberately builds upon that foundation. However, while effective in unconstrained environments, the multi-agent nature introduces significant architectural challenges when enforcing real-world limitations, since many constraints span multiple agents and cannot be addressed independently. Consequently, this thesis extends and adapts the multi-agent framework to enable coordinated, global, constraint-aware decision-

making.

1.2 Scientific relevance

The academic relevance of this project lies in its contribution to both research and practice. From a research perspective, it addresses a gap in the application of DRL to infrastructure planning. While the use of DRL in this domain is gaining traction, it is still in its early stages, with most of the key literature having emerged only in the last decade. Moreover, real-world constraints and heuristics are rarely integrated into these models. Most academic work either ignores them entirely or handles them through simplistic mechanisms like reward shaping, which limit robustness and generalizability. This thesis contributes by advancing the use of methodologically grounded constraint-handling techniques that are better suited for complex decision spaces and have broader relevance beyond this case.

Moreover, while many DRL studies rely on toy problems or abstract simulations, this research is rooted in a real-world, high-dimensional case using validated municipal data. The case of Amsterdam's quay walls is not only practically important but also technically challenging. Because recent literature highlights the immense promise of multi-agent architectures for managing complex, multi-component infrastructure networks, this study deliberately adopts a multi-agent DRL approach. However, integrating strict constraints and heuristics into such systems remains a research frontier due to the added complexity of inter-agent coordination and ensuring global feasibility. As such, the thesis contributes not only by adapting and testing advanced methods but also by bringing DRL closer to practical deployment in infrastructure management. It serves as a bridge between theory and implementation, and between state-of-the-art algorithms and the operational realities faced by asset managers.

This thesis addresses these challenges by developing a multi-agent DRL framework that explicitly incorporates constraints and domain-specific heuristics, enabling maintenance planning for urban infrastructure that balances long-term efficiency with practical feasibility in real-world settings.

1.3 Management of Technology (MOT) Relevance

This thesis aligns closely with the objectives of the Management of Technology (MOT) programme. It presents a scientific study in a technological context, applying DRL to support strategic maintenance planning for critical urban infrastructure. The research treats advanced decision-support models as a corporate resource, aimed at improving long-term planning, risk management, and cost efficiency under real-world constraints. By focusing on how asset owners can use data-driven methods to coordinate interventions, manage uncertainty, and balance competing objectives, the study adopts a clear managerial and organizational perspective. Methodologically, the thesis applies scientific modeling, simulation, and optimization techniques consistent with the MOT curriculum, demonstrating how

technology-enabled decision-making can enhance organizational performance and infrastructure reliability.

1.4 Research objectives

Building on the motivation and scientific relevance outlined above, this research aims to translate the identified challenges into concrete goals. The main objectives of this thesis are:

- Identify and mathematically classify real-world constraints and domain-specific heuristics relevant to urban infrastructure interventions, with a focus on Amsterdam's historic quay walls.
- Evaluate state-of-the-art constraint- and heuristic-handling techniques in DRL and determine their suitability for integration into complex, multi-agent systems.
- Develop a constraint- and heuristic-aware DRL framework based on a multi-agent architecture capable of coordinating interventions under shared constraints and heuristics.
- Benchmark the developed framework against rule-based baselines and unconstrained DRL to evaluate the impact of integrating constraints and heuristics on the performance of DRL-generated maintenance policies.

1.5 Research questions

To guide the scope and focus of this investigation, the following research questions were formulated:

Main Research Question

> *How can real-world constraints and domain-specific heuristics be effectively integrated into DRL models to generate practical, scalable, and cost-effective maintenance policies for complex infrastructure systems?*

Sub-Research Questions

1. What types of real-world constraints and domain-specific heuristics are most relevant to urban infrastructure maintenance planning, and which of these should be treated as hard constraints (strictly enforced), soft constraints (penalized but flexible), or heuristic guidance (informal but useful)?

2. Which DRL constraint- and heuristic-handling methods are best suited for integrating the identified constraints and heuristics in the context of complex infrastructure maintenance planning?

3. How can inter-agent constraints and heuristics be effectively enforced in a Multi-agent DRL framework, and how can the scalability of the training process be facilitated for larger infrastructure networks?

4. How well does the proposed DRL framework perform in generating valid and efficient intervention plans in terms of constraint and heuristic compliance, cost-effectiveness, disruption minimization, and asset reliability?

5. How does the constraint- and heuristic-aware DRL method perform compared to rule-based baselines and unconstrained DRL.

2 Theoretical Background

This section presents the theoretical foundations underlying the methodology adopted in this thesis. As introduced earlier, the maintenance planning framework is based on DRL, which provides a flexible and scalable approach to sequential decision-making under uncertainty. DRL is particularly well suited to managing deteriorating multi-component infrastructure systems, as it can handle stochastic deterioration processes, long planning horizons, and large state and action spaces without requiring a fully specified analytical model of the system.

The section begins with Markov Decision Processes (MDPs), the core mathematical framework of reinforcement learning, and extends to Partially Observable Markov Decision Processes (POMDPs) to capture the limited and uncertain information typical in real-world infrastructure systems. Building on this foundation, RL and DRL are introduced, emphasizing how neural network function approximation enables learning in complex, high-dimensional environments.

The discussion then focuses on actor–critic methods as the class of DRL algorithms employed in this work, followed by Multi-Agent Reinforcement Learning (MARL), motivating the use of a DCMAC architecture to manage the multi-component nature of the quay wall system. These concepts establish the theoretical basis for the constrained optimization approaches presented in the subsequent chapters.

2.1 Markov Decision Processes (MDPs)

MDPs constitute the fundamental theoretical framework underlying DRL. An MDP provides a formal mathematical model of the interaction between an agent and its environment, describing decision-making under uncertainty through states, actions, rewards, and state transitions [17].

Formally, an MDP is fully characterized by the tuple $\langle S, A, P, R, \gamma \rangle$, where:

- S denotes a finite set of system states,
- A represents a finite set of admissible actions,
- P is the state transition probability function,
- R defines the immediate reward associated with state–action pairs, and
- γ is the discount factor that weights future rewards.

At each discrete decision epoch t, the agent observes the current state s_t, selects an action a_t, receives an immediate reward R_{t+1}, and transitions to a subsequent state s_{t+1} according

to the probabilistic dynamics defined by P (see Figure 1). A central assumption underlying MDPs is the Markov Property, which states that the probability distribution of the next state depends solely on the current state and selected action, and is independent of the prior history of states and actions.

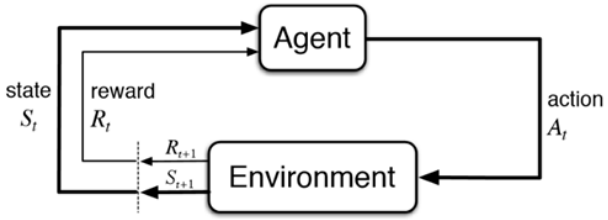

Figure 1: Agent–Environment interaction in an MDP [1]

The agent's behavior over time is described by a policy π, which specifies how actions are chosen. Policies may be deterministic, mapping each state to a single action, or stochastic, assigning a probability distribution over actions for each state.

The objective of the agent is to learn an optimal policy π^* that maximizes the expected long-term return, defined as the discounted cumulative reward (Equation (1)):

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right] \tag{1}$$

To evaluate policies and guide decision-making, MDPs employ value functions, which quantify the expected future reward associated with states and actions. The two principal value functions are:

- **State-value function** (Equation (2)): the expected return when starting from state s and following policy π.

$$V^{\pi}(s) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s \right] \tag{2}$$

- **Action-value function (Q-value)** (Equation (3)): the expected return obtained by executing action a in state s, and thereafter following policy π.

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a \right] \tag{3}$$

For any MDP, there exists at least one optimal policy π^* that simultaneously maximizes the expected return from every state. This optimal policy induces the optimal state-value

function $V^*(s)$ and optimal action-value function $Q^*(s, a)$, defined as the supremum over all admissible policies (Equation (4)).

$$\begin{aligned} V^*(s) &= \max_{\pi} V^{\pi}(s) \\ Q^*(s, a) &= \max_{\pi} Q^{\pi}(s, a) \end{aligned} \tag{4}$$

These optimal value functions satisfy the Bellman optimality equations, which provide recursive characterizations of optimal behavior and form the theoretical foundation of most reinforcement learning and deep reinforcement learning algorithms. In practice, exact computation of $V^*(s)$ and $Q^*(s, a)$ becomes intractable for large or continuous state–action spaces, motivating the use of approximate methods and function approximation techniques that are central to DRL [18, 19].

2.2 Partially Observable Markov Decision Processes (POMDPs)

While MDPs assume that the agent has full access to the environment's current state, many real-world problems involve partial observability, where the agent cannot directly observe the true underlying state. Instead the agent has to use other data like data from sensors, deterioration models, etc., to estimate the state in which the environment is [20]. To formally model decision-making under such uncertainty, POMDPs extend the MDPs framework by incorporating a notion of observations.

A POMDP is defined by the tuple $\langle S, A, P, R, \gamma, \Omega, O \rangle$, where the first five elements are identical to those in an MDP, and the additional components are:

- Ω – a finite set of possible observations the agent can receive,
- O – an observation probability function that defines the likelihood of observing $o \in \Omega$ given the environment is in state s' and the agent executes action a.

At each decision epoch t, the agent does not directly observe the true state s_t, but instead receives an observation o_t drawn according to O. The agent must then choose an action based on its history h_t(Equation (5)) of past actions and observations.

$$h_t = (o_0, a_0, o_1, a_1, \ldots, o_t) \tag{5}$$

Due to partial observability, policies in POMDPs are generally defined over belief states $b(s)$, which are probability distributions over the underlying states, representing the agent's internal estimate of the environment, as shown in Equation (6):

$$b_t(s) = P(s_t = s \mid h_t) \tag{6}$$

The agent's goal is to learn a belief-based policy $\pi(b)$ that maps beliefs to actions in order to maximize the expected discounted return (Equation (7)):

$$\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid b_0\right] \tag{7}$$

Analogous to MDPs, POMDPs employ value functions, but now defined over beliefs:

- **Belief-state value function** $V^{\pi}(b)$ (Equation (8)): expected return when starting from belief b and following policy π.

$$V^{\pi}(b) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid b_0 = b\right] \tag{8}$$

- **Belief-action value function** $Q^{\pi}(b, a)$ (Equation (9)): expected return of taking action a in belief b and thereafter following policy π.

$$Q^{\pi}(b, a) = \sum_{o \in \Omega} P(o \mid b, a) V^{\pi}(b') \tag{9}$$

POMDP-based formulations have been widely applied to infrastructure inspection and maintenance problems [21, 22], often in combination with point-based solution methods [23, 24, 25, 26, 27]. These studies demonstrate that POMDPs can effectively capture complex uncertainty and decision-making under partial observability, frequently outperforming heuristic or rule-based strategies. However, their practical applicability is constrained by severe computational challenges, particularly in environments with large or continuous state and action spaces. Exact solutions are generally intractable because the belief space is continuous and high-dimensional.

Consequently, most real-world implementations rely on approximate methods, including belief state discretization, point-based value iteration, or model-free RL approaches that operate directly on observations and histories. These approximations enable RL algorithms to tackle partially observable real-world problems, including infrastructure maintenance problems, where full state information is rarely available.

2.3 Reinforcement Learning (RL)

The dependency on explicitly defining probabilistic models for the environment can be alleviated using RL techniques, where the decision maker, the so-called agent, interacts directly with the environment with the objective of learning optimal decision-making policies. While the underlying system dynamics still implicitly govern state or belief transitions, RL can operate even when these transition probabilities are unknown or difficult to estimate.

RL methods can broadly be classified into model-based and model-free approaches. Model-based algorithms, such as Dynamic Programming (DP) with value or policy iteration, require knowledge of the state transition probabilities and expected rewards for all state-action pairs to compute the optimal policy. In contrast, model-free methods—including Temporal Difference (TD) learning, SARSA, and Q-learning—derive optimal solutions by directly interacting with the environment, using observed samples of state transitions and rewards rather than a predefined model.

Another distinction concerns on-policy versus off-policy learning. In on-policy approaches (e.g., SARSA), the agent updates its value estimates based on actions generated by the current policy π, seeking to maximize the expected sum of discounted rewards. Off-policy methods (e.g., Q-learning) allow the agent to learn the optimal policy π^* while exploring the environment using different or even random actions, enabling more flexible exploration strategies.

Despite their theoretical appeal, classical RL techniques encounter significant limitations in high-dimensional or combinatorial state- and action-spaces. Tabular methods, such as Q-learning, become computationally infeasible as the number of states and possible actions grows, limiting their practical applicability to complex real-world systems.

2.4 Deep Reinforcement Learning (DRL)

A transformative advancement in RL occurred in 2014, when DeepMind demonstrated that combining RL with deep neural networks could achieve human-level performance in Atari video games [28]. This approach, known as DRL, replaces explicit tabular representations of value functions with neural network approximations (Figure 2). The network parameters θ serve as a compact proxy for the underlying value functions, enabling generalization across large or continuous state spaces while significantly reducing computational requirements [29].

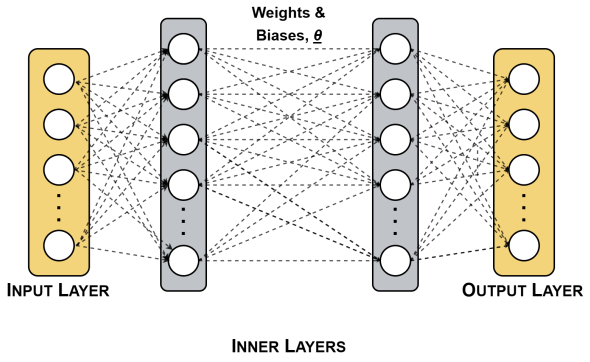

Figure 2: Neural Network diagram

These capabilities make DRL particularly well-suited for complex decision-making prob-

lems, such as the maintenance planning of the Amsterdam quay walls. The environment involves numerous interacting components, stochastic degradation dynamics, and long-term dependencies between actions and outcomes. Such complexity would be difficult to manage with classical RL or exact POMDP methods alone.

2.4.1 Actor-critic DRL

Within DRL, algorithms can be broadly categorized into value-based methods and policy-gradient methods, depending on how the optimal policy is learned. Value-based methods learn an optimal policy implicitly by estimating value functions (e.g., state- or action-value functions) and selecting actions that maximize the expected return. In contrast, policy-gradient methods learn an optimal policy explicitly by directly optimizing the parameters of a stochastic policy to maximize the expected cumulative reward.

A hybrid approach that combines the advantages of both paradigms is provided by actor–critic algorithms, which simultaneously learn a policy (the actor) and a value function (the critic) [30]. The actor represents the policy $\pi(a|s;\theta)$ and selects actions, while the critic evaluates these actions by estimating a value function, such as $V(s;\phi)$ or $Q(s,a;\phi)$. The critic supplies a learning signal—typically a Temporal-Difference (TD) error or an advantage estimate—that guides the actor's policy updates.

The general architecture of actor–critic methods is illustrated in Figure 3. When applied in a DRL setting, neural networks are used to approximate both the actor and critic, enabling learning in high-dimensional or continuous state and action spaces. This DRL version of the actor–critic architecture is shown in Figure 4.

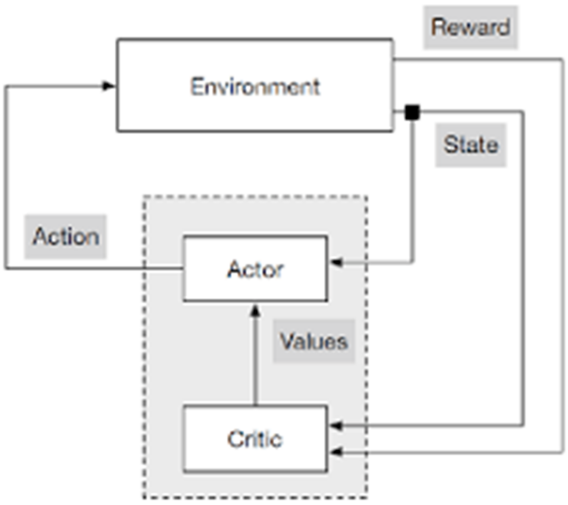

Figure 3: Actor-Critic Architecture [2]

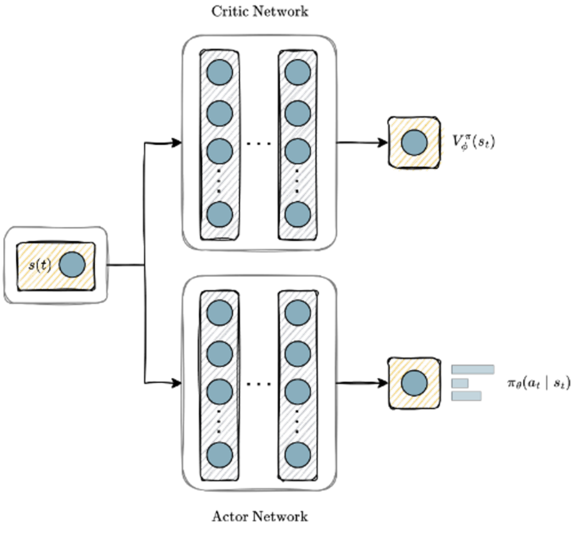

Figure 4: DRL Actor-Critic Architecture

By combining policy optimization with value-based evaluation, actor–critic methods achieve lower variance in policy-gradient updates while maintaining the ability to operate in complex, high-dimensional environments. This makes them particularly suitable for the quay wall maintenance problem. For these reasons, an actor–critic approach forms the core DRL method employed in this thesis.

2.5 Architecture: Multi-Agent Deep Reinforcement Learning

A key challenge in applying RL to large-scale infrastructure systems is the curse of dimensionality: as the number of agents (or system components) increases, the joint state and action spaces grow exponentially, rendering centralized single-agent approaches computationally intractable. An important advantage of actor–critic methods, however, is that they naturally extend to Multi-Agent Reinforcement Learning (MARL) settings. In MARL, multiple agents learn simultaneously while interacting within a shared environment, where system dynamics may depend on the collective actions of all agents.

A common strategy to mitigate the dimensionality of large joint action spaces in MARL is to assume conditional independence of component-level actions given the global state. Under this assumption, the joint policy can be factorized, as shown in Equation (10).

$$\pi(a_1, \ldots, a_N \mid s) = \prod_{i=1}^{N} \pi_i(a_i \mid s) \tag{10}$$

a_i denotes the action of agent i at time t. This factorization decomposes the joint action vector into a product of individual policies, thereby avoiding the combinatorial explosion associated with enumerating all possible joint action combinations and substantially reducing computational complexity.

In this thesis, a multi-agent extension known as Deep Centralized Multi-Agent Actor Critic (DCMAC) is employed [14]. This architecture is characterized by the presence of multiple actors—one per system component (in this case, one per quay wall)—that select actions independently, while a centralized critic evaluates the collective effect of all actions during training. The critic leverages global state and joint action information to provide a consistent learning signal, enabling coordinated behavior across agents.

For the Amsterdam quay wall case study, each quay wall is modeled as an agent with five possible states and five possible maintenance actions. The MARL formulation therefore learns a policy that maps the global system state, encompassing the condition of all quay walls, to an optimal joint action, specifying one action per wall at each decision step.

Although DCMAC employs multiple actors (Figure 5), in contrast to fully decentralized approaches such as Deep Decentralized Multi-Agent Actor-Critic (DDMAC) (Figure 6) [15], all actors in the DCMAC architecture share a common actor network. This parameter sharing enforces consistency across agents, improves sample efficiency, and facilitates scalability, while

still allowing each component to act independently based on its local role within the global system.

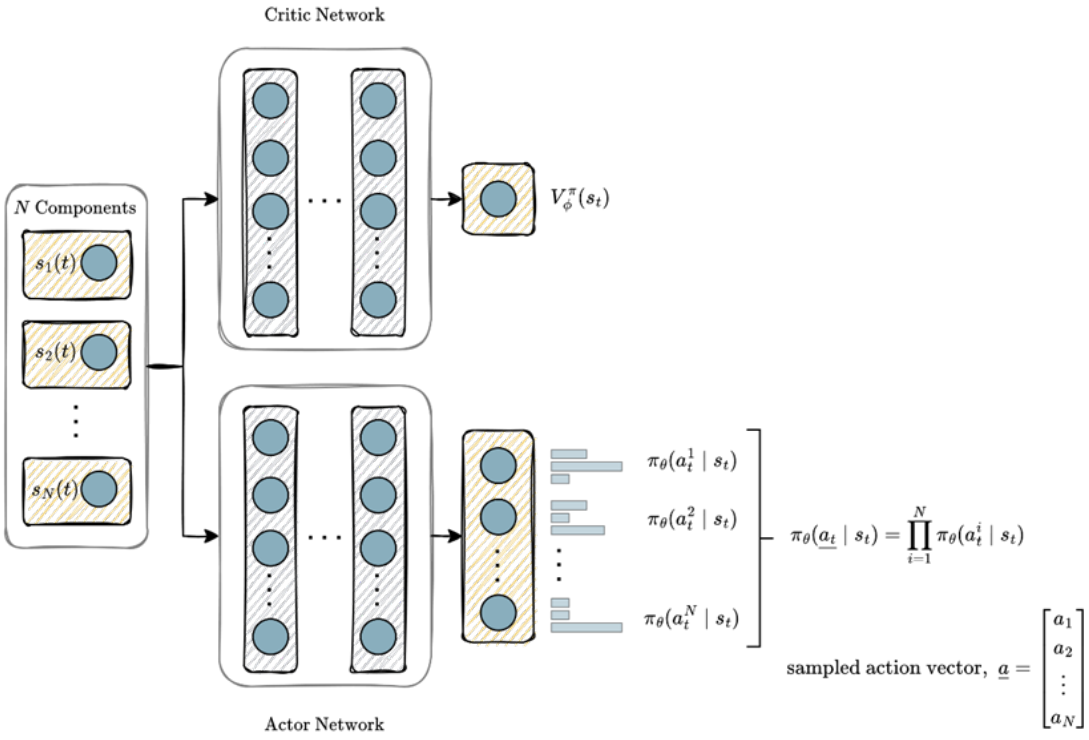

Figure 5: DCMAC Architecture

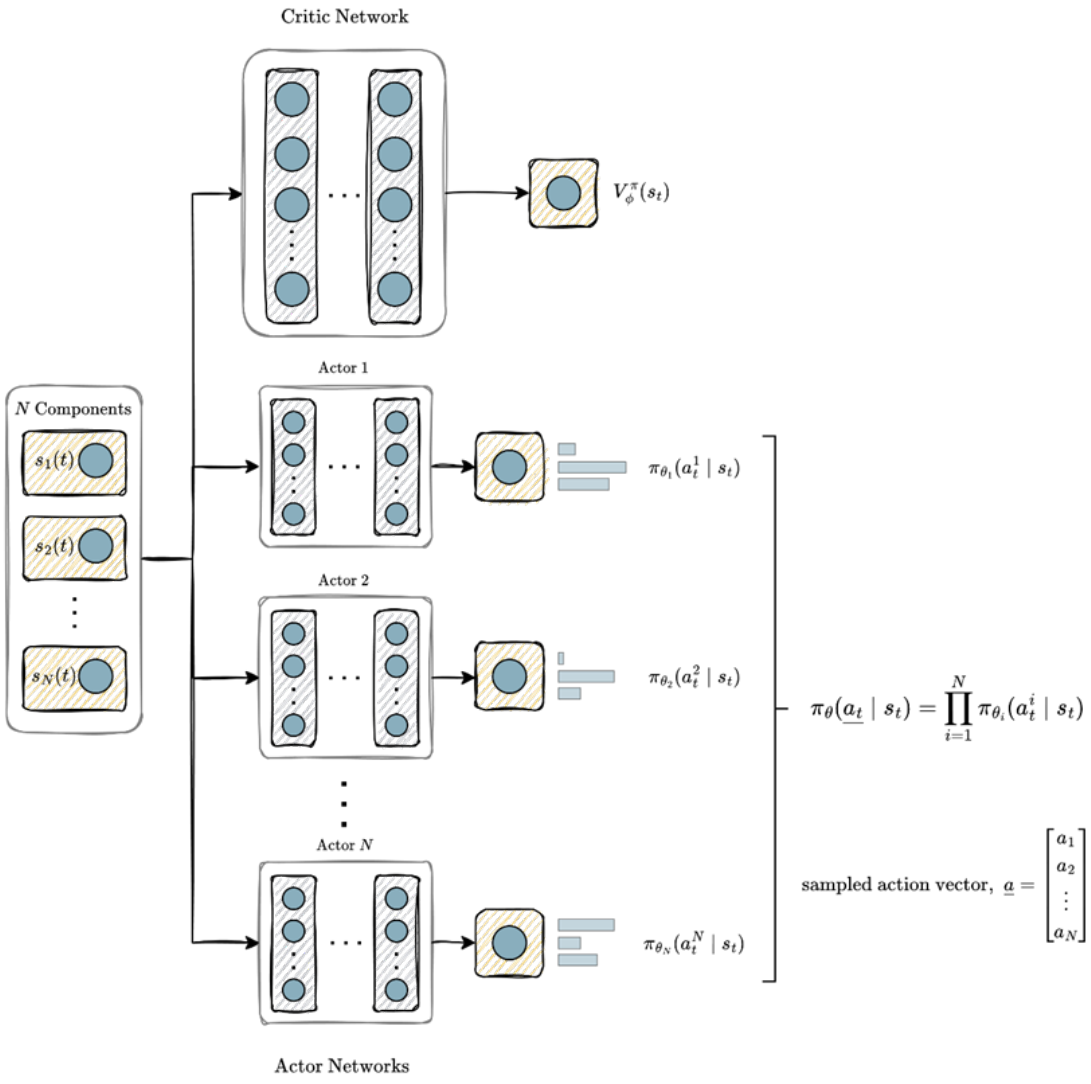

Figure 6: DDMAC Architecture

Previous studies have demonstrated that DCMAC can achieve significant life-cycle cost reductions in infrastructure maintenance applications [14], highlighting its suitability for large-scale, multi-component decision-making problems such as quay wall maintenance planning.

3 Case study and problem definition

This chapter presents the real-world case study used to train and evaluate the proposed DRL algorithm for optimal maintenance planning and formally defines the associated decision-making problem. The case study focuses on the network of historic quay walls in the city of Amsterdam, a critical urban infrastructure system facing increasing structural risks due to ageing, growing loads, and complex interactions with the urban environment.

The chapter first introduces the physical and historical context of Amsterdam's quay walls and outlines their structural typologies. It then describes the municipal risk assessment methodology, which forms the basis for defining the discrete deterioration states used in this study. Building on this, the deterioration process, maintenance actions, transition dynamics, and cost structure are modelled within an MDPs framework.

To capture network-level societal impacts, the chapter additionally introduces the adopted city cost modelling approach based on the US digital twin, which quantifies traffic and environmental effects of maintenance interventions. Finally, all elements are integrated into a formal MDP formulation, defining the state space, action space, transition dynamics, and reward function.

Finally, building directly upon these physical and economic definitions, all elements are integrated into a formal Markov Decision Process (MDP). This formulation explicitly translates the domain knowledge into the state space, action space, transition dynamics, and reward function required by the learning algorithm. Together, these components fully specify the optimisation problem addressed by the DRL agent and provide the necessary context for the methodologies and results presented in subsequent chapters.

3.1 Quay walls in Amsterdam

A quay wall is a vertical retaining structure constructed along the edge of a harbour or river to retain soil and provide a stable interface for berthing, loading, and unloading vessels [3].

In Amsterdam, quay walls are an integral part of the historic canal system and constitute a vital element of the city's infrastructure. The earliest quay walls in the city date back to the thirteenth century and were constructed exclusively from timber. With the development of new construction materials and advances in engineering knowledge [31, 32], alternative structural solutions emerged. The earliest documented use of stone quay walls in Amsterdam dates to the sixteenth century [33]. Although quay wall construction techniques evolved over time, the predominant structural typology of Amsterdam's historic quay walls (Figure 7A) originates from designs that began to be widely adopted in the Netherlands during the seventeenth century [34].

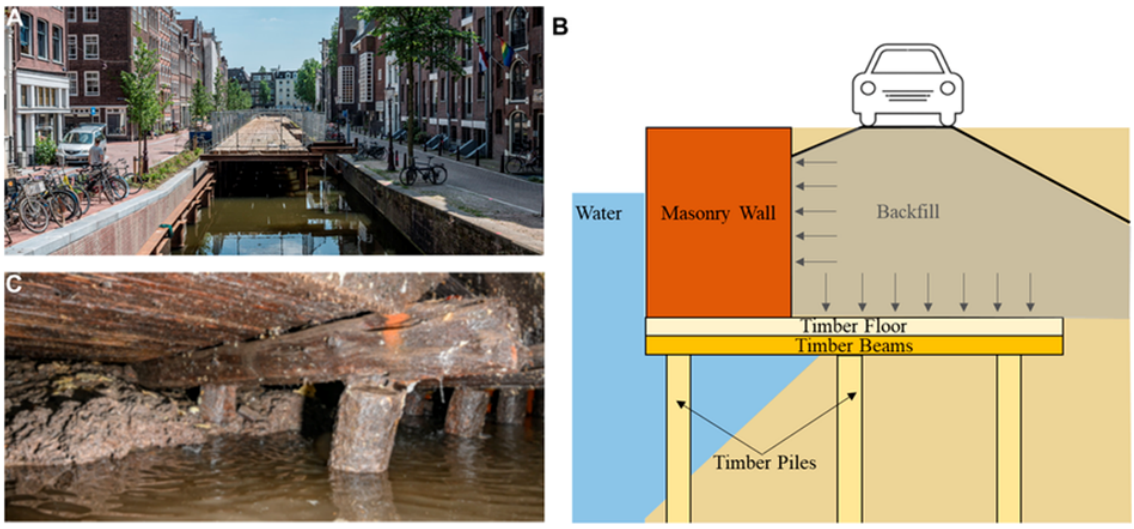

Figure 7: Architecture schematic of Amsterdam's historic quay walls [3].

A schematic of a cross section of such quays can be seen in Figure 7B. A masonry wall is placed on a timber floor against the backfill with a foundation of timber piles. In between the foundation piles and the floor, timber beams called *kespen* are placed (Figure 7C). The timber piles support the whole system and have a geotechnical function: they are driven to reach the first layer of strong soil [3].

Despite their historical origin, quay walls remain a high priority in the city's infrastructure agenda. Over centuries, the timber piles supporting these structures have gradually deteriorated, leading to a reduction in structural capacity. At the same time, traffic intensity and associated loads on the quay walls have increased significantly. Together, these factors contribute to a rising probability of structural failure. In response to these challenges, the Municipality of Amsterdam initiated a large-scale renovation and repair programme in 2020, known as *Programma Bruggen en Kademuren*, targeting more than 200 km of quay walls. The objective of this programme is to ensure safe and accessible quay walls both in the present and in the future [35].

3.2 Environment Overview

The simulation environment considers a planning horizon of 20 years with annual time steps. The case study focuses on the Prinsengracht area in Amsterdam, comprising 70 quay wall segments grouped into 26 quay wall units, with a total length of 4,424.42 meters.

A quay wall unit (referred to as a *group* in this project) is defined as a continuous section of quay wall located between two bridges on one side of a canal. Each unit may consist of multiple segments (referred to as *components*), reflecting partial replacements carried out at different points in time. To ensure consistent tracking within the simulation and optimization framework, each quay wall group and each individual component is assigned a

unique identifier. Figure 8 presents a map of the studied area and highlights the quay wall units included in the analysis.

Figure 8: Map of studied quay walls in Amsterdam (Generated via US Tool).

The initial condition state of each segment is provided by the Municipality of Amsterdam and is determined using the ARK assessment methodology.

3.3 Structural integrity and risk assessment of quay walls

To systematically assess the risk of failure of inner-city quay walls, the Municipality of Amsterdam has developed a methodology known as ARK (*Amsterdamse Risicobeoordeling Kademuren*) [36]. The purpose of ARK is to evaluate structural risks in a consistent and transparent manner, enabling informed decision-making regarding inspection, maintenance, and intervention priorities.

The ARK methodology combines information from archival records, visual inspections, measurements, and other data sources to assign safety scores to various aspects of a quay

wall's condition. These aspects include, among others, the age of the structure, the condition of the foundation, and the stability of the riverbed. In parallel, ARK assesses the potential consequences of failure by considering factors such as land use above the quay wall, usage of the adjacent waterway, and the height of the masonry wall.

The assessed failure probability and consequence scores are each classified on a scale from 1 to 4. These scores are multiplied to obtain an overall risk score, which is used to support maintenance and renewal decisions. Table 1 presents the relationship between probability of failure and consequence scores as defined within the ARK framework.

Score	State / Consequence
≤ 1.0	Low
≤ 2.0	Middle
≤ 3.0	High
> 3.0	Very high

Table 1: Probability of failure / consequence scores according to ARK

Building upon the ARK methodology, this project adopts the same classification system to define discrete condition states for each quay wall: *green*, *light orange*, *dark orange*, and *red*. To better align with the objectives of this study and to explicitly account for collapse-related costs, an additional terminal state, *failed*, is introduced. This state represents structural failure and is required to capture the costs incurred once a quay wall reaches a critical condition. Table 2 summarizes the set of possible states used in the model.

State / Condition
Good Condition
Medium Condition
Bad Condition
Very Bad Condition
Failed

Table 2: Possible state for each Quay Wall of the system

The primary objective of the model is to estimate the likelihood of quay walls reaching the *failed* state over their lifetime. This requires modelling how the condition of a quay wall evolves over time as it transitions between ARK-defined states. This sequence of state changes is referred to as the deterioration process.

3.4 Maintenance actions and transition dynamics

To manage the deterioration process, the Municipality of Amsterdam can adopt a range of strategies, including structural interventions, load reduction measures, or postponing action. In this study, five maintenance actions are considered:

- Do nothing
- Close road for parking
- Close road for traffic
- Lifetime extension
- Full renovation

Due to the inherent uncertainty in material degradation, environmental effects, and loading conditions, the underlying physical deterioration of the assets is a stochastic process. Consequently, transitions between condition states are probabilistic.

The state transition dynamics are represented using transition matrices (Tables 3 to 7). In these matrices, rows correspond to the current state, columns to the subsequent state, and the matrix entries represent transition probabilities.

For passive actions (Tables 3 to 5), transitions are restricted to equal or more deteriorated states, with different deterioration rates depending on the action. Under the *Do nothing* action (Table 3), transition probabilities between consecutive states follow a geometric progression, with each subsequent transition probability doubling. This parametrisation results in an expected lifespan of approximately 100 years from the initial green state to failure.

Do nothing		**Next state**				
						Failed
Current state		98.13%	1.87%	-	-	-
		-	96.25%	3.75%	-	-
		-	-	92.5%	7.5%	-
		-	-	-	85%	15%
	Failed	-	-	-	-	100%

Table 3: Transition matrix for "Do nothing"

The mitigation actions *Close road for parking* (Table 4) and *Close road for traffic* (Table 5) follow the same geometric pattern but with slower deterioration rates, extending the expected lifespan to 150 and 200 years, respectively.

Close parking		Next state				
						Failed
Current state		98.75%	1.25%	-	-	-
		-	97.5%	2.5%	-	-
		-	-	95%	5%	-
		-	-	-	90%	10%
	Failed	-	-	-	-	100%

Table 4: Transition matrix for "Close parking"

Close traffic		Next state				
						Failed
Current state		99.06%	0.94%	-	-	-
		-	98.13%	1.87%	-	-
		-	-	96.25%	3.75%	-
		-	-	-	92.5%	7.5%
	Failed	-	-	-	-	100%

Table 5: Transition matrix for "Close traffic"

Active maintenance actions yield deterministic outcomes. A *lifetime extension* (Table 6) improves the condition state by one level (except for the failed state), while a *full renovation* (Table 7) restores the quay wall to the *green* state regardless of its previous condition.

All active maintenance actions are assumed to take exactly one year to complete. This assumption was made to align seamlessly with the discrete one-year timestep used throughout the simulation environment, keeping the Markov Decision Process (MDP) structurally straightforward. While in reality a full renovation or a lifetime extension might take more or less than twelve months depending on the specific site, aggregating the intervention duration to a single timestep is a standard and practical simplification for long-term strategic planning. If a higher temporal resolution is ever required for operational planning, the framework can easily accommodate this by simply reducing the timestep duration (e.g., to months or quarters) and scaling the deterioration transition probabilities accordingly.

Lifetime extension		Next state				
						Failed
Current state		100%	-	-	-	-
		100%	-	-	-	-
		-	100%	-	-	-
		-	-	100%	-	-
	Failed	-	-	-	-	100%

Table 6: Transition matrix for "Lifetime extension"

Full renovation		Next state				
						Failed
Current state		100%	-	-	-	-
		100%	-	-	-	-
		100%	-	-	-	-
		100%	-	-	-	-
	Failed	100%	-	-	-	-

Table 7: Transition matrix for "Full renovation"

It should be emphasized that the transition matrices used in this study represent a simplified approximation of the true deterioration process. This simplification introduces uncertainty that propagates through the decision-making framework. For real-world implementation, these matrices would require calibration based on structural reliability analyses and historical performance data of quay walls.

3.5 Cost structure

In addition to the transition dynamics, the model incorporates deterministic cost matrices defined for each combination of current state, action, and next state. Three cost components are considered: direct maintenance costs, failure-related costs, and city costs representing broader urban impacts. Together, these components balance short-term intervention costs against long-term system reliability.

3.5.1 Direct maintenance costs

Maintenance costs are proportional to quay wall length. A lifetime extension intervention costs €25,000 per meter, while full renovation costs €50,000 per meter. These values were established in collaboration with the Municipality of Amsterdam and are applied uniformly across all segments.

3.5.2 Failure costs

Failure costs are incurred only when a segment reaches the *failed* state. The cost per linear meter is determined by the ARK consequence score (*gevolg score*, ranging from 1 to 4) using the expression in Equation (11):

$$C_{\text{failure}} = 50,000 \cdot (\text{Score}_{\text{consequence}} - 1). \tag{11}$$

This formulation ensures that failures in high-consequence areas result in proportionally higher penalties. The calculation captures the broader societal impact of failure, while direct reconstruction expenses are already included in the direct maintenance costs.

3.5.3 City costs

City costs represent annual societal impacts associated with maintenance actions and are derived using US, the digital twin developed by TNO. These costs account for traffic disruption and environmental impacts, quantified using the following monetary values:

- Traffic disruption: €15 per vehicle loss hour
- Environmental impacts:
  - CO_2 emissions: €100 per metric ton
  - NO_2 and NO_x emissions: €27 per kg
  - PM2.5: €484 per kg

– PM10: €50 per kg

The total city cost is calculated by combining these factors with reference parameters (a wall height of 3 m and a city surface area of 220 km^2) and scaling peak-hour simulation outputs to annual values.

In practice, the raw city costs computed using US are considerably smaller in magnitude than the direct maintenance and failure-related costs. To ensure that urban hindrance plays a meaningful role in the agent's decision-making process, a scaling factor of 10 is applied to these city costs. Without this adjustment, financial costs would completely dominate the optimization process, and the agent would mostly ignore the societal impact of traffic disruption. This scaling is a deliberate modeling choice driven by three main reasons.

First, the US tool strictly monetizes vehicle travel delays and specific exhaust emissions. However, true urban disruption in a city like Amsterdam goes much further. Prolonged street closures trigger a ripple effect: local businesses lose revenue, emergency services face delays, noise pollution increases, and cyclists and pedestrians are heavily impacted. The scaling factor acts as a necessary proxy to capture these unquantified societal burdens.

Second, monetizing societal impact is inherently subjective. Pinning the cost of a lost hour in traffic at €15 is just one estimate; a municipality might decide to penalize public hindrance much more heavily depending on their political goals. Just as failure costs depend on municipal accounting rules, the weight assigned to city disruption is ultimately a flexible policy lever.

Finally, from a methodological standpoint, a primary goal of this thesis is to test whether the DRL agent can navigate complex trade-offs. To prove that the agent can genuinely balance financial limits against city-wide disruptions, these competing objectives must be somewhat comparable in magnitude. It is much more robust to train and validate a model in an environment where it is forced to actively account for significant city disruptions. If a municipality later decides to give societal impact less weight, the agent will still easily find an optimal solution for that simpler environment. Conversely, if the model were trained with negligible city costs, the agent would never learn how to navigate those complex network interactions, and might fail to find an optimal policy if the municipality suddenly decided to increase the weights in the future.

Further details on the computation of city costs using US and their integration into the optimization framework are provided in Section 3.7.

3.6 Time-value of money

The objective of the optimisation problem is to identify the sequence of actions that minimises the lifetime costs of each component, given its current condition and stochastic deterioration dynamics. To account for the time value of money, including inflation and

opportunity costs, future costs are discounted using a factor of 0.99 per year. Consequently, costs incurred one year later are valued at 99% of their nominal value.

3.7 City cost modelling using Urban Strategy

3.7.1 Motivation and network-level effects

The objective of this research is to derive optimal maintenance policies at the network level of quay walls rather than for individual assets in isolation. This network perspective is essential because maintenance actions introduce coupling effects through urban hindrance. For example, the traffic impact caused by simultaneously closing two adjacent streets is generally not equal to the sum of the impacts of closing each street independently. Capturing such non-linear interactions is crucial when evaluating maintenance strategies in a dense urban environment such as Amsterdam.

To quantify these network-wide effects, city-related costs associated with maintenance actions are estimated using traffic and environmental simulations. These simulations are performed with the Urban Strategy (US) Tool, developed by TNO, which enables fast evaluation of large numbers of traffic scenarios through highly parallelized computations [37].

3.7.2 Urban Strategy as a digital twin framework

US is a digital twin platform designed to support policy analysis and infrastructure planning in urban environments. It integrates transport models with environmental impact modules, allowing the assessment of mobility, air quality, and related indicators within a single computational framework. The platform is specifically optimized for speed through parallel algorithms and GPU-based computation, enabling simulations that typically require hours in traditional tools to be completed within minutes.

In this case study, US is built upon the VMA 4.0, which provides the underlying road network and Origin-Destination matrices (Figure 9). These data serve as the foundation for all subsequent simulations.

From the base transport model, multiple US modules are employed:

- **Traffic+**, which assigns car and freight traffic to the road network based on link capacities and travel speeds;
- **Air**, which estimates changes in air pollutant emissions resulting from altered traffic patterns;
- **TrafIndic**, which aggregates simulation outputs into indicators suitable for policy evaluation.

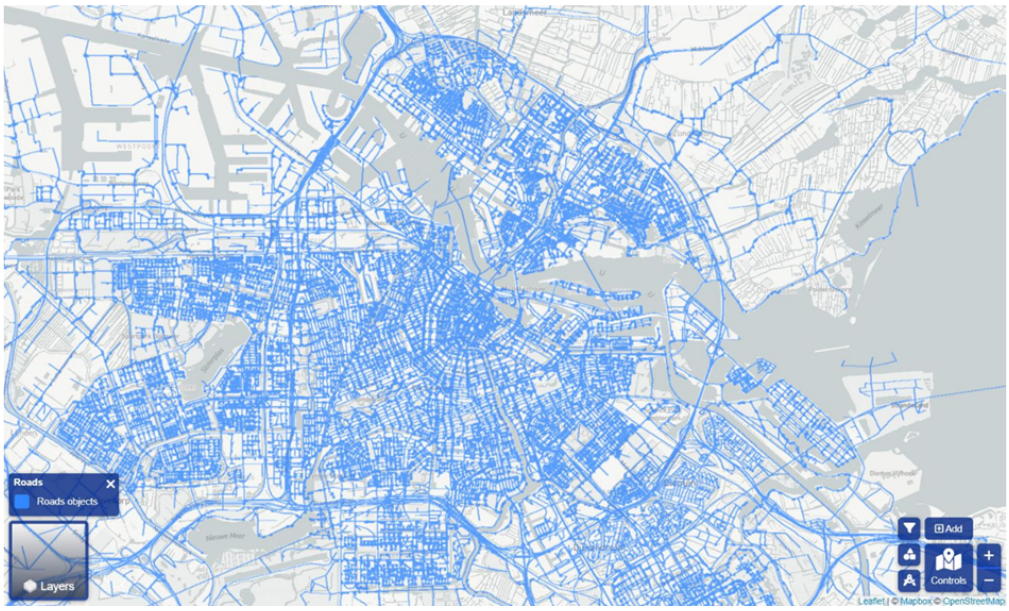

Figure 9: Screenshot of VMA 4.0 road network in US.

Within Traffic+, freight traffic is assigned using an all-or-nothing approach, assuming drivers always choose the shortest path. Passenger car traffic is distributed using an equilibrium-based assignment (Volume Averaging), allowing congestion effects to influence route choice. The Air module translates traffic flows into emissions of CO_2, NO_2/NO_x, $PM_{2.5}$, and PM_{10}, while TrafIndic computes indicators such as vehicle loss hours and total vehicle kilometres travelled.

3.7.3 Representation of maintenance-related interventions

For the quay wall maintenance problem, US is used to evaluate the city-level impacts of interventions that affect traffic conditions but do not directly alter the structural state of the quay wall. Three traffic-related actions are considered within US:

- Do nothing,
- Close road for parking,
- Close road for traffic.

The latter two actions generate city costs due to increased travel times, congestion, and environmental impacts. Closing a road for traffic is implemented directly in US by setting the road's capacity and speed to zero, effectively removing the road from the usable network.

Closing a road for parking is not available as a standard intervention in the tool. To approximate its effect, the road speed is reduced by 50%, making the link less attractive to

drivers. This modelling choice reflects the assumption that removing parking reduces traffic searching for parking spaces and discourages through traffic on the affected street.

In the context of maintenance planning, any maintenance action involving construction work is assumed to require a full street closure. Consequently, whenever a maintenance action (e.g., lifetime extension or full renovation) is performed on a quay wall section, the corresponding quay wall unit is modelled as being closed to traffic during the intervention period. This assumption reflects common practice in Amsterdam, where construction activities along quay walls typically necessitate closing the adjacent road to all forms of traffic.

US simulations are executed for a representative two-hour morning peak period. The resulting changes in traffic flows and emissions are subsequently scaled to annual values and converted into monetary terms using predefined weighting factors. This procedure allows city costs to be expressed in euros, making them directly comparable with maintenance and failure costs in the optimization framework.

3.7.4 Calculation of Societal Interaction Costs

Precomputation strategy

While US enables relatively fast simulations (averaging around a minute per simulation), directly embedding traffic simulations within the evaluation loop of a DRL algorithm is computationally infeasible. The number of policy evaluations required during training would result in an impractical runtime. To understand why, one must consider the sheer scale of policy evaluations required during training. Deep reinforcement learning algorithms like PPO learn through extensive trial and error, typically requiring millions of interactions with the environment to converge to a stable policy. If the US tool were triggered on the fly at every single decision step, a single training run would require tens of millions of minutes—equating to several years of continuous computation. Given that this research required executing hundreds of training runs for hyperparameter tuning, constraint testing, and model validation, online simulation is entirely impractical.

To address this limitation, all US simulations are performed offline in a precomputation phase. The resulting city costs are stored in a lookup table that maps combinations of street closures to their associated societal costs. During optimization, the DRL agent retrieves the relevant city costs from this table rather than triggering new simulations.

However, the number of possible closure combinations grows exponentially with the number of quay wall units. In the present case study, 26 quay wall units are considered, which would theoretically result in over 2.5 trillion possible closure combinations. Exhaustively simulating all configurations is therefore impossible.

To make the precomputation tractable, the number of simultaneous closures is limited. In this study, US simulations are therefore performed for all configurations in which at most

three quay wall units are closed at the same time. This process is carried out separately for the two intervention types: closing roads for traffic and closing roads for parking.

In total, this results in approximately 5,900 unique simulations, requiring around 98 hours of computation time. The output of these simulations is stored in a CSV file containing city costs for all combinations of up to three closures per intervention type. The methodological implications of this modeling decision, are discussed in detail in Section 3.7.5 and further addressed in the future work recommendations in Chapter 9.

Estimation of city costs during simulation

Once the offline precomputation phase has been completed, the resulting database of city costs is used during the DRL simulation to evaluate the societal impact of maintenance decisions without invoking additional US simulations. At each decision step, the agent selects an action for every quay wall component, resulting in a system-wide action vector. This action vector is translated into city costs through a two-step procedure formalized in Algorithms 1 and 2.

Algorithm 1 Calculation of City Costs

Require: Action vector *action_vector*
Initialize *total_cost* $\leftarrow 0.0$
Initialize empty sets *groups_close_parking* and *groups_close_traffic*
for each action in *action_vector* **do**
    Retrieve *group_id* of the group quay wall corresponding to the action
    if action is *"close traffic"* or *"lifetime extension"* or *"full renovation"* **then**
        Add *group_id* to *groups_close_traffic*
    else if action is *"close parking"* **then**
        Add *group_id* to *groups_close_parking*
    end if
end for
for each *group_id* in *groups_close_parking* **do**
    if *group_id* $\in$ *groups_close_traffic* **then**
        Remove *group_id* from *groups_close_parking*
    end if
end for
Calculate costs for *groups_close_parking* using **Algorithm 2** and add to *total_cost*
Calculate costs for *groups_close_traffic* using **Algorithm 2** and add to *total_cost*

Algorithm 2 Calculation of City Costs for Group Combinations

Require: *group_ids* set and type of action (*close parking* or *close traffic*)
Ensure: *combined_cost*
1: Initialize *combined_cost* $\leftarrow 0.0$
2: Sort *group_ids* in ascending order
3: Split the sorted *group_ids* into batches of up to 3 groups
4: **for** each batch **do**
5: Retrieve the stored cost for the corresponding group combination and *action_type*
6: Add the retrieved cost to *combined_cost*
7: **end for**

3.7.5 Limitations and implications

The adopted approach provides a computationally feasible way to incorporate network-level hindrance effects into the DRL framework. Nevertheless, it introduces simplifications that may affect the accuracy of the estimated city costs. Partitioning closures into fixed subsets of three and ordering them by group identifier does not fully capture spatial interactions between quay wall units. For instance, four spatially clustered closures may have a substantially different impact than the sum of three clustered closures plus one isolated closure, yet both situations may be treated similarly by the approximation.

These limitations can influence the learned policies of the DRL agent, particularly in scenarios involving many simultaneous interventions. Addressing this issue represents a key opportunity for future work. Potential improvements include alternative aggregation strategies, increased coverage of precomputed combinations, or the use of surrogate models that explicitly account for spatial proximity and network topology. These directions are further discussed in the future research chapter (Chapter 9).

3.8 Formalisation of the MDP

To integrate the case study described above into the DRL framework, the problem is formulated as an MDP. This requires specifying the state space, action space, transition dynamics, and reward function in a manner consistent with the real-world quay wall maintenance problem.

3.8.1 State space

The environment consists of N quay wall components (segments), indexed by $i \in \{1, \ldots, N\}$.

The local state of component i at time t is denoted by $s_i^t \in \mathcal{S}_i$, where the discrete states correspond to ARK-based condition classes:

$$\mathcal{S}_i = \{0 \text{ (Green)}, 1 \text{ (Light Orange)}, 2 \text{ (Dark Orange)}, 3 \text{ (Red)}, 4 \text{ (Failed)}\}.$$

The global system state at time t is defined as the Cartesian product of all component states (Equation (12)).

$$s_t = (s_1^t, s_2^t, \ldots, s_N^t) \in S^N \tag{12}$$

This formulation enables the agent to reason at the network level while preserving component-level deterioration dynamics.

3.8.2 Action space

At each decision step t, the agent selects a maintenance action for each component. The local action for component i is denoted by $a_i^t \in \mathcal{A}_i$, corresponding to the following set of actions:

$$\mathcal{A}_i = \{0 \text{ (Do nothing)}, 1 \text{ (Cl. parking)}, 2 \text{ (Cl. traffic)}, 3 \text{ (Lif. ext.)}, 4 \text{ (Full Ren.)}\}.$$

The joint action applied to the system at time t is defined in Equation (13).

$$a_t = (a_1^t, a_2^t, \ldots, a_N^t) \in A^N \tag{13}$$

This decentralized action structure reflects the fact that maintenance decisions are taken at the level of individual quay wall segments, while their consequences may interact at the level of quay wall units and the surrounding urban network.

3.8.3 Transition dynamics

State transitions are governed by stochastic deterioration processes that depend on the current state and the selected action. For each component i, the transition probability is defined as:

$$P(s_i^{t+1} \mid s_i^t, a_i^t)$$

with action-specific transition matrices defined as described in Section 3.4. Assuming conditional independence between components, the global transition probability factorizes as shown in Equation (14).

$$P(s_{t+1} \mid s_t, a_t) = \prod_{i=1}^{N} P(s_i^{t+1} \mid s_i^t, a_i^t) \tag{14}$$

This assumption allows for scalable simulation while preserving the impact of joint actions through the reward function.

3.8.4 Reward function and objective

The objective of the agent is to learn an optimal policy π^* that maximizes the expected discounted cumulative reward (Equation (15)):

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{T} \gamma^t R_t(s_t, a_t) \right] \tag{15}$$

where:

- γ is the discount factor reflecting the time value of money.
- T denotes the length of the planning horizon.

The immediate reward at time t is defined as the negative of the total system cost (Equation (16)):

$$R_t(s_t, a_t) = -\frac{1}{\delta} \left(C^t_{\text{direct}} + C^t_{\text{failure}} + C^t_{\text{city}} \right) \tag{16}$$

where δ is a scaling factor introduced to reduce numerical magnitudes (originally on the order of millions of euros) and to improve numerical stability during DRL training.

Direct maintenance costs Direct maintenance costs represent immediate intervention expenses and are additive over components (Equation (17)):

$$C^t_{\text{direct}} = \sum_{i=1}^{N} c_i^{\text{direct}}(a_i^t) \tag{17}$$

where:

- N is the total number of components (quay wall segments);
- $c_i^{\text{direct}}(a_i^t)$ denotes the direct maintenance cost incurred for component i when action a_i^t is selected, which, as described in Section 3.5, depends on the length of component i and the chosen maintenance action.

Failure costs Failure costs capture the expected societal impact of structural collapse and are modelled as risk-weighted costs (Equation (18)):

$$C^t_{\text{failure}} = \sum_{i=1}^{N} P_{\text{fail}}(s^t_i, a^t_i) \cdot c^{\text{failure}}_i \tag{18}$$

where:

- N is the total number of components;
- $P_{\text{fail}}(s^t_i, a^t_i)$ is the probability that component i transitions to the failed state when action a^t_i is applied in state s^t_i;
- c^{failure}_i is the failure cost associated with component i, determined by the ARK consequence score of the corresponding quay wall.

City (interaction) costs City costs represent network-level societal impacts resulting from traffic and parking disruptions. Unlike direct and failure costs, these costs are not additive over individual components but depend on the joint configuration of actions across quay wall units (Equation (19)):

$$C^t_{\text{interaction}} = \sum_{k \in \{\text{parking,traffic}\}} C^{\text{group}}_k(G^t_k) \tag{19}$$

where:

- $k \in \{\text{parking}, \text{traffic}\}$ denotes the type of intervention;
- G^t_k is the set of quay wall group identifiers affected by intervention type k at time t;
- $C^{\text{group}}_k(G^t_k)$ is the combined city cost associated with the simultaneous closure of the groups in G^t_k under intervention type k.

As explained in **Algorithm 1** and **Algorithm 2**, group assignment follows the rules:

$$G^t_{\text{parking}} = \{g_i \mid a^t_i = \text{“close parking”}\} \setminus G^t_{\text{traffic}},$$

$$G^t_{\text{traffic}} = \{g_i \mid a^t_i \in \{\text{“close traffic”}, \text{“lifetime extension”}, \text{“full renovation”}\}\}.$$

For a set of affected groups $G = \{g_1, g_2, \ldots, g_m\}$, the combined group cost is defined as (Equation (20)):

$$C^{\text{group}}_k(G) = \begin{cases} \text{lookup}(G, k), & |G| \leq 3 \\ \sum_{j=1}^{\lceil m/3 \rceil} \text{lookup}(G_j, k), & |G| > 3 \end{cases} \tag{20}$$

where:

- G_j denotes the j-th subset of at most three groups obtained by sorting G by group identifier;
- lookup$(\cdot, k)$ retrieves the precomputed city cost from the Urban Strategy–based cost table described in Section 3.7.

3.9 Configuration of the DRL Agent

To address the optimisation problem defined by the MDP formulation, this study utilizes the Proximal Policy Optimization (PPO) algorithm, a state-of-the-art Actor-Critic DRL method. This approach employs neural networks to approximate both the policy (the Actor) and the value function (the Critic), enabling efficient learning in the high-dimensional state space of the quay wall network while ensuring stable policy updates.

The specific architecture designed for this case study (DCMAC) is presented in Figure 10. The network input is a representation of the global environment state s_t, encompassing the current belief states for all N quay wall components.

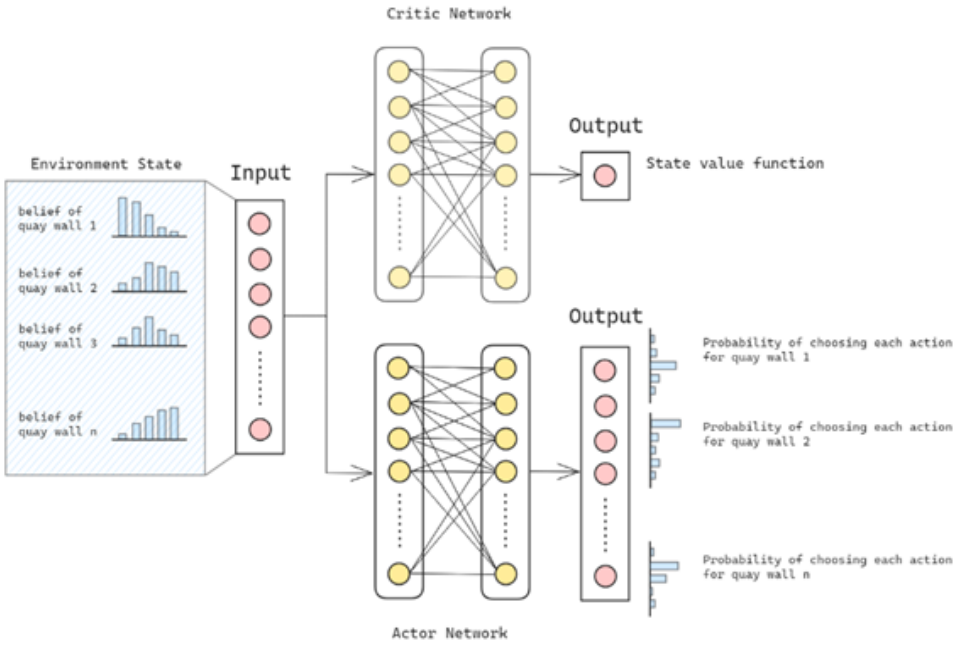

Figure 10: Actor-Critic network architecture

The architecture processes this state through two distinct subnetworks:

- **The Critic Network**: Estimates the state-value function, outputting a single scalar value that assesses the expected long-term reward from the current global state. This estimate is used to critique the actions taken by the actor during training.

- **The Actor Network**: Determines the control policy. Reflecting the component-level definition of the action space defined in Equation (13), the Actor's output layer is structured to provide separate probability distributions over the five available maintenance actions for each individual quay wall component.

4 Methodology I: Unconstrained DRL Framework

Having formally defined the maintenance of Amsterdam's quay walls as a Markov Decision Process (MDP) in the previous chapter, the focus now shifts to the computational framework employed to solve it. As established in the theoretical background, the high dimensionality of the state space—resulting from the combinatorial explosion of component conditions and network-level interactions—renders exact dynamic programming methods intractable. Consequently, this research leverages DRL to approximate the optimal maintenance policy.

This chapter details the implementation of the core learning framework used to address the base optimization problem, prior to the integration of complex operational constraints. The objective is to establish a robust, stable, and effective learning engine capable of navigating the stochastic deterioration dynamics and cost structures defined in the case study. It is important to note that the foundation of this unconstrained framework builds upon previous research developed at TNO, which established the core DRL architecture [38]. Following extensive debugging of the original codebase, targeted architectural modifications, and rigorous hyperparameter tuning, this adapted model served as the unconstrained baseline for the remainder of this thesis.

The methodology is grounded in the use of PPO, a state-of-the-art actor-critic algorithm selected for its stability and sample efficiency. The first section of this chapter provides a theoretical derivation of PPO, detailing the transition from standard policy gradients to the clipped surrogate objective that enables reliable learning in complex environments. Following this, the specific adaptation of the MDP for a finite-horizon setting is addressed. Since the quay wall maintenance plan covers a fixed 20-year period, the standard Markov assumption must be reinforced through state augmentation, explicitly incorporating temporal information to allow the agent to learn horizon-dependent strategies.

This chapter focuses exclusively on the design and training of the unconstrained agent. The rigorous validation of this framework, including benchmarking against heuristic baselines and mathematical optimization algorithms, is reserved for Chapter 5, which establishes a dedicated validation framework. Subsequently, the extension of the DRL agent to handle hard and soft constraints is presented in Chapter 6.

4.1 Proximal Policy Optimization (PPO)

Policy-gradient methods in RL were often suffering from instability, frequently because large policy updates could unexpectedly harm performance. To address this, trust-region methods were introduced to limit how much the policy is allowed to change at each update, ensuring learning remains reliable. The first widely adopted algorithm following this idea was Trust Region Policy Optimization (TRPO) [39], which constrains updates by bounding the Kullback-Leibler (KL) divergence between successive policies (Equation (21)), leading to

more stable and consistent policy improvement.

$$\begin{aligned} &\max_{\theta} \mathbb{E}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} A_t \right] \\ &\text{s.t. } \mathbb{E}_t \left[\text{KL} \left(\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \right) \right] \leq \delta \end{aligned} \tag{21}$$

However, the constrained nature of TRPO, combined with its limited flexibility for mini-batch updates, makes it computationally expensive and slow. To address these issues, Schulman et al. proposed Proximal Policy Optimization (PPO) [40], which retains the stability benefits of TRPO but replaces the constrained optimization with a simple clipped objective, allowing multiple gradient updates per batch. As a result, PPO is simpler, faster, easier to tune, and has become the more widely used policy-gradient algorithm in practice.

4.1.1 Clipped Surrogate Objective

PPO controls policy updates by limiting how much the new policy can differ from the old one using a clipped objective. If we define the probability ratio using Equation (22), the standard policy gradient objective is given by Equation (23):

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{22}$$

$$\mathbb{E}_t \left[r_t(\theta) \, A_t \right] \tag{23}$$

As mentioned previously, directly maximizing Equation (23) can lead to overly large and unstable policy updates. PPO instead maximizes the clipped objective shown in Equation (24):

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \ \text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) A_t \right) \right] \tag{24}$$

Here, ϵ is a small constant (e.g., 0.2). The clipping removes incentives for excessive policy changes by capping improvements when $r_t(\theta)$ leaves the interval $[1 - \epsilon, 1 + \epsilon]$. This results in a conservative objective that stabilizes learning while remaining simple and efficient to optimize.

4.1.2 PPO Algorithm

As a result, the clipped surrogate objective leads to a practical and efficient learning procedure, commonly summarized by Algorithm 3 (from [7]). At a high level, PPO alternates between collecting trajectories using the current policy and performing multiple gradient updates on the policy and value function using the same batch of data.

Algorithm 3 Proximal Policy Optimization (PPO) [7]

Require: Initial policy parameters θ_0, value function parameters ϕ_0
Require: Number of iterations K, clipping parameter ϵ
1: **for** $k = 0, 1, \ldots, K-1$ **do**
2: Collect set of trajectories $\mathcal{D}_k = \{\nu_i\}$ by running policy π_{θ_k} in the environment
3: Estimate returns G_t
4: Estimate advantages A_t using the value function V_{ϕ_k}
5: Update policy parameters by maximizing:

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_t \left[\min \left(r_t(\theta) A_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right]$$

6: Update value function parameters by regression:

$$\phi_{k+1} = \arg\min_{\phi} \mathbb{E}_t \left[\left(V_\phi(s_t) - G_t \right)^2 \right]$$

7: **end for**

While this base algorithm captures the core idea of PPO, practical implementations require several additional modifications to achieve stable and reliable performance in complex environments.

4.1.3 Practical PPO Implementation Details

In practice, PPO is almost never implemented exactly as described in its base formulation. A comprehensive overview of the most important implementation choices is provided by Huang et al. [41], who identify 37 details that materially affect PPO performance. The most relevant of these modifications for this thesis are discussed below.

I. Exploration vs. Exploitation

PPO learns a stochastic policy in an on-policy setting, meaning actions are sampled from the current policy distribution rather than chosen deterministically. This inherent stochasticity provides a natural mechanism for balancing exploration and exploitation: early in training, high policy entropy encourages exploration, while later updates gradually shift probability mass toward higher-reward actions.

However, policy-gradient updates alone may still lead to premature convergence and insufficient exploration. To mitigate this, practical implementations often include an entropy bonus, which encourages the policy to maintain a degree of randomness and continue exploring. Incorporating this term, the total objective that PPO seeks to maximize becomes the one shown in Equation (25):

$$L_{\text{Critic}}(\theta) = L^{\text{CLIP}}(\theta) + c_2 L_{\text{entropy}}(\theta) \tag{25}$$

Here, $L_{\text{entropy}}(\theta)$ is the expected policy entropy, and c_2 is a scaling coefficient that balances exploration against policy improvement. By including this term, the agent is incentivized

not only to maximize expected returns but also to avoid overly confident or deterministic policies prematurely, supporting more robust learning.

II. Advantage Calculation

PPO relies on an estimate of the advantage function A_t, which quantifies how much better an action is compared to the average behavior of the policy at a given state. The simplest approach to compute the advantage is to use the one-step Temporal Difference (TD) error (Equation (26)):

$$A_t^{\mathrm{TD}} = R_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{26}$$

where R_t is the reward at time t, $\gamma \in [0, 1]$ is the discount factor, and V_ϕ is the value function. While this estimator is simple and efficient, it is biased because it only looks one step ahead and ignores long-term rewards.

Generalized Advantage Estimation (GAE) [42] improves on the one-step TD error by considering multiple steps into the future while controlling variance through an exponentially decaying factor $\tau \in [0, 1]$. The GAE advantage is computed as a weighted sum of future TD errors (Equation (27)):

$$A_t^{\mathrm{GAE}(\gamma,\tau)} = \sum_{l=0}^{\infty} (\gamma\tau)^l \, \delta_{t+l} \tag{27}$$

where δ_t, given by Eq. (28), is the TD error at time t:

$$\delta_t = R_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{28}$$

The parameter τ controls the trade-off between bias and variance:

- $\tau = 0$ reduces to the standard one-step TD advantage (low variance, high bias);
- $\tau = 1$ approximates the full Monte Carlo advantage (low bias, high variance).

By using a weighted sum of multiple TD errors, GAE provides a more accurate and stable advantage estimate, which improves both policy gradient updates and overall learning efficiency in PPO, making it the most widely used method to compute the advantage in practice.

III. Advantage Normalization

In addition to GAE, it is common practice to normalize advantages within each batch before performing policy updates. Given a batch of advantages $A_{t=1}^T$, normalized advantages are computed as defined in Equation (29):

$$\hat{A}_t = \frac{A_t - \mu_A}{\sigma_A}, \tag{29}$$

where μ_A and σ_A denote the mean and standard deviation of the advantages in the batch.

Advantage normalization does not change the expected gradient direction but significantly improves numerical stability by preventing large-magnitude advantages from dominating the update. Empirically, this leads to more consistent learning dynamics, faster convergence, and reduced sensitivity to reward scaling, which is especially important in environments with heterogeneous or high-variance cost structures.

IV. Probability Ratios via Log-Probabilities

Although the PPO objective is defined in terms of probability ratios, practical implementations compute these ratios using log-probabilities (Equation (30)):

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \rightarrow r_t(\theta) = \exp\left(\log \pi_\theta(a_t \mid s_t) - \log \pi_{\theta_{\text{old}}}(a_t \mid s_t)\right). \tag{30}$$

This formulation is numerically more stable, particularly when action probabilities are very small, and avoids underflow issues that can arise when directly dividing probabilities. Moreover, modern deep learning frameworks compute log-probabilities efficiently and accurately, making this approach standard in PPO implementations.

4.1.4 Implemented PPO Framework

Incorporating the considerations discussed above, the PPO variant used in this thesis aligns more closely with practical, modern implementations of PPO than with the original formulation.

At a high level, training proceeds in iterative cycles: rollout data is collected, returns and normalized advantages are computed, and both the actor and critic networks are updated via minibatch gradient descent (Algorithms 4 to 6).

Algorithm 4 Training of the Agent

Require: Hyperparameters
Initialize policy (actor) network weights θ
Initialize value function (critic) network weights ϕ
for $i = 1, \ldots,$ train_iters **do**
    $t \leftarrow 0$, $s_t \leftarrow$ reset environment
    for $j = 1, \ldots,$ steps_per_iter **do**
        $t \leftarrow t + 1$
        $\pi_\theta(\cdot \mid s_t) \leftarrow \text{ActorNet}(s_t)$ # forward pass of actor network
        $V_\phi(s_t) \leftarrow \text{CriticNet}(s_t)$ # forward pass of critic network
        $a_t \leftarrow$ sample from $\pi_\theta(\cdot \mid s_t)$
        $\pi_\theta(a_t \mid s_t) \leftarrow$ evaluate probability of a_t under $\pi_\theta(\cdot \mid s_t)$
        $s_{t+1}, R(s_t, a_t) \leftarrow$ environment step
        Store tuple $\{s_t, a_t, \pi_\theta(a_t \mid s_t), V_\phi(s_t), R(s_t, a_t)\}$ in Rollout Memory D_k
        $s_t \leftarrow s_{t+1}$
        if $t = T$ **or** $j =$ steps_per_iter **then** # Returns and advantages are computed after full pass through the environment
            if $t = T$ **then**
                $V_\phi(s_{t+1}) \leftarrow 0$
                $s_t \leftarrow$ reset environment
            else
                $V_\phi(s_{t+1}) \leftarrow \text{CriticNet}(s_{t+1})$
            end if
            $\delta_t \leftarrow R(s_t, a_t) + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
            Compute returns: $G_t \leftarrow \sum_{l=0}^{T-t-1} \gamma^l R_{t+l}$
            Compute advantages: $A_t \leftarrow \sum_{l=0}^{T-t-1} (\gamma\tau)^l \delta_{t+l}$
            Store δ_t, A_t in D_k
        end if
    end for
    Train Agent (D_k) using Algorithm 5 # training of networks
end for

Algorithm 5 Training of the Networks

Require: Rollout Memory $D_k \rightarrow$ shuffled and divided into minibatches (over multiple epochs)

1: **Update Critic Network:** Update parameters ϕ, using the Critic loss function:

$$L_{\text{Critic}}(\phi) = \frac{1}{T} \sum_{t=1}^{T} \left(V_\phi(s_t) - G_t\right)^2$$

2: **Update Actor Network:** Update parameters θ, using the Actor loss function with entropy regularization:

$$L_{\text{Actor}}(\theta) = \sum_{t=1}^{T} \left[\min\Big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\Big) + c_2\, H[\pi_\theta(\cdot \mid s_t)] \right]$$

where

$$r_t(\theta) = \exp\Big(\log \pi_\theta(a_t \mid s_t) - \log \pi_{\theta_{\text{old}}}(a_t \mid s_t)\Big) \quad \text{(computed following \textbf{Algorithm 6})}$$

$$\hat{A}_t = \frac{A_t - \mu_A}{\sigma_A} \text{ is the normalized advantage at timestep } t$$

$$H[\pi_\theta(\cdot \mid s_t)] = -\sum_a \pi_\theta(a \mid s_t) \log \pi_\theta(a \mid s_t)$$

Algorithm 6 Computation of PPO Probability Ratio $r_t(\theta)$

Require: Rollout Memory D_k

Require: Current policy parameters θ

1: Retrieve s_t, a_t and $\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ from rollout memory D_k

2: Compute current policy distribution:

$$\pi_\theta(\cdot \mid s_t) \leftarrow \text{ActorNet}_\theta(s_t)$$

3: Compute the probability of the taken action a_t under $\pi_\theta(\cdot \mid s_t)$:

$$\pi_\theta(a_t \mid s_t)$$

4: Compute probability ratio using log-probabilities:

$$r_t(\theta) \leftarrow \exp\Big(\log \pi_\theta(a_t \mid s_t) - \log \pi_{\theta_{\text{old}}}(a_t \mid s_t)\Big)$$

Collectively, these algorithms define the PPO training pipeline employed throughout this work and provide the foundation for all experimental results presented in the following chapters.

4.2 Problem Modeling and State Augmentation

The quay wall maintenance problem considered in this thesis is modeled as a Markov Decision Process (MDP), defined by the tuple:

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$$

At each decision step t, the agent observes a state s_t, selects an action a_t, receives a reward r_t, and transitions to a new state s_{t+1} according to the transition dynamics $\mathcal{P}(s_{t+1}|s_t, a_t)$. A core assumption of this framework is the Markov property: the state must contain all information required to make optimal decisions about the future, as it defines a memoryless stochastic process that depends solely on the current state and not the sequence of events that preceded it.

In the quay wall maintenance setting, the state naturally includes the structural condition of the quay walls. However, structural information alone is insufficient for the objective considered here. The goal is to minimize total maintenance costs over a finite planning horizon of 20 years, which induces a non-stationary optimal policy. Actions that are optimal early in the horizon—such as costly renovations with long-term benefits—may become suboptimal near the end of the horizon, where the remaining time is insufficient to realize their value.

If temporal information is omitted, two situations with identical structural conditions but different remaining horizons become indistinguishable to the agent. As a result, the agent is forced to select the same action in both cases, violating the Markov property with respect to the finite-horizon objective and impeding effective learning.

To address this issue, the state representation needs to be extended through state augmentation. State augmentation refers to the inclusion of additional variables in the state that are necessary for optimal decision-making but are not captured by the base environment state. Formally, an augmented state is defined as shown in Equation (31).

$$\tilde{s}_t = (s_t, z_t) \tag{31}$$

Where s_t denotes the original environment state and z_t represents auxiliary contextual information. The policy is then conditioned on the augmented state, $\pi(a_t|\tilde{s}_t)$, enabling the agent to exploit this additional information when selecting actions.

4.2.1 Temporal State Augmentation

The choice of augmentation depends on the problem objective. In stationary settings with an infinite planning horizon, the original MDP formulation is sufficient, and the optimal policy depends only on the structural state. In contrast, the quay wall maintenance problem is explicitly finite-horizon and therefore non-stationary. Without access to time, the agent

cannot distinguish between early and late stages of the planning horizon and cannot learn horizon-dependent trade-offs.

This limitation is particularly problematic in the final years of the horizon, where it may be optimal to accept higher failure risks rather than perform expensive interventions whose benefits extend beyond the planning period. To restore the Markov property under a finite-horizon objective, time is explicitly included in the state (Equation (32)).

$$\tilde{s}_t = (s_t, t) \tag{32}$$

With this formulation, the agent is aware of its position within the planning horizon, and the resulting MDP satisfies the Markov property for the optimization problem considered. This enables the learning of time-dependent policies that appropriately balance short-term and long-term costs.

In practice, the time variable is normalized to promote stable and efficient learning. The augmented state is therefore defined as in Equation (33):

$$\tilde{s}_t = \left(s_t, \frac{t}{T}\right), \tag{33}$$

where T denotes the total planning horizon. Expressing time as a fraction of the horizon ensures scale consistency with other state variables, improves numerical stability, and encourages policies that generalize across different horizon lengths.

4.2.2 Rationale for the Augmented Formulation

For the remainder of this thesis, the time-augmented MDP is adopted for the following reasons:

- Public infrastructure planning is inherently constrained by finite budget cycles and administrative mandates, making finite-horizon optimization more relevant for stakeholders such as the Municipality of Amsterdam.
- While a finite horizon risks the agent neglecting maintenance in the final years to save costs, explicitly tracking time can also counteract this behavior. It provides the framework with the temporal awareness needed to enforce end-of-horizon condition constraints, ensuring the long-term health of the assets.
- The inclusion of a discount factor γ reflects the economic time value of money; without temporal information, the agent cannot correctly exploit this structure.
- Temporal information is required to correctly formulate and interpret additional constraints introduced later in this thesis.

- Finally, as a supplementary methodological benefit, formulating a finite-horizon optimization problem allows for a fair, direct comparison with the Cross-Entropy Method (CEM), as both approaches can be evaluated under the exact same temporal boundaries.

5 Methodology II: Validation Framework

The validation of the developed model is a critical part of this thesis. To assess its performance, the results obtained with the proposed DRL agent will be benchmarked against alternative approaches throughout the thesis. In particular, comparisons with rule-based and reactive strategies are included, as these methods reflect current practice in infrastructure maintenance planning. While such benchmarking is useful to demonstrate improvements over existing approaches, it is not sufficient to validate the model itself, as outperforming heuristic or rule-based methods does not guarantee that the learned policy is close to optimal.

Proper validation requires assessing whether the model is capable of reaching an optimal, or near-optimal, solution. For the real-world problem considered in this thesis, however, this is not directly verifiable. The problem's dimensionality is extremely large, reflecting the complexity of urban infrastructure systems with many interacting components, uncertain deterioration, and long planning horizons. As discussed in the introduction, no known mathematical optimization or exact solution method can feasibly solve this problem at full scale. As a result, there is no ground-truth optimal solution against which the DRL model can be directly compared.

To address this limitation, validation will be carried out using a simplified toy environment with substantially reduced dimensionality. In this smaller setting, the number of components, states, and actions is limited such that a classical mathematical optimization algorithm can be applied to compute an optimal (or close to optimal) solution. This makes it possible to directly evaluate whether the DRL agent is able to learn a policy that matches, or closely approximates, the true optimum.

In a second step, the same mathematical optimization approach will be applied to a larger environment, after which its performance and scalability will be compared to that of the developed DRL model. This comparison is intended to empirically demonstrate the curse of dimensionality: as problem size increases, generic mathematical optimization methods become computationally infeasible, while the DRL approach remains applicable and effective.

Taken together, this validation strategy serves two purposes. First, it verifies that the developed DRL model is capable of finding near-optimal solutions. Second, it provides evidence that DRL is a suitable and scalable approach for high-dimensional infrastructure maintenance problems, where traditional optimization techniques fail to scale and cannot be applied in practice.

5.1 Baseline Policies

To evaluate the effectiveness of the proposed DRL agent, its performance is compared against several alternative strategies that reflect existing practice in infrastructure maintenance planning. First, a set of heuristic policies is considered. These policies are designed to

emulate the decision-making of an experienced asset manager: they rely solely on the current condition of each quay wall and deterministically map each state to a corresponding action. The heuristic policies evaluated in this work are summarized in Table 8.

	Good	Medium	Bad	Very Bad	Failed
Reactive to failure	Do nothing	Do nothing	Do nothing	Do nothing	Full renovation
Reactive to critical condition	Do nothing	Do nothing	Do nothing	Life extension	Full renovation
Mitigation of critical condition	Do nothing	Do nothing	Do nothing	Close traffic	Full renovation
Gradual response	Do nothing	Close parking	Close traffic	Life extension	Full renovation
Early intervention	Do nothing	Close traffic	Life extension	Full renovation	Full renovation

Table 8: Heuristic policies considered

In addition to the heuristics, the DRL agent is benchmarked against results from a previous TNO project, in which maintenance actions for each quay wall were determined individually using MDPs [43]. While this MDP formulation provides a useful reference, it does not account for interactions between quay-wall units. Consequently, it neglects the impact of network-level dependencies and aggregated city costs that arise when multiple components are maintained simultaneously.

Despite this limitation, the MDP-based approach serves as a meaningful benchmark, allowing us to quantify the effect of considering city-level interactions on the total costs obtained by the DRL agent. Comparing the DRL results with both heuristic and MDP-based benchmarks highlights the importance of explicitly modeling interdependencies within the quay-wall network to achieve cost-effective maintenance planning.

5.2 Cross Entropy Method (CEM)

The Cross-Entropy Method (CEM) was developed by P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein in the late 1990s [44]. Although it was not originally intended as a general-purpose optimization technique, it later evolved into a powerful method for solving problems traditionally addressed by heuristic search algorithms. In particular, it has been successfully applied to policy search in MDPs, where it is capable of finding high-quality policies efficiently [45].

In this report, we apply the CEM to determine the optimal action sequence $[a_0, a_1, \ldots, a_{T-1}]$, which maximizes the expected return, as defined in Equation (34).

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T-1} \gamma^t \, R(s_t, a_t) \;\middle|\; S_0 = s\right] \tag{34}$$

In this section, we present a structured description of the CEM as applied to sequential decision-making in MDPs. We begin by motivating the choice of CEM as the underlying optimization approach, emphasizing its suitability as a derivative-free, simulation-based benchmark. We then provide a conceptual and mathematical overview of the method, framing

trajectory optimization as a probabilistic estimation problem. Building on this foundation, we describe the standard CEM algorithmic framework and subsequently introduce a set of enhancements designed to improve stability, exploration, and convergence efficiency. Finally, we present the complete algorithm as implemented in this work, incorporating all proposed modifications and parameter schedules.

5.2.1 Motivation for using CEM

The CEM was selected for its simplicity, flexibility, and proven effectiveness in tackling complex decision-making problems where classical optimization methods are either impractical or computationally expensive. As a derivative-free, simulation-based algorithm, CEM does not require gradient information or assumptions about convexity, continuity, or linearity. This makes it well-suited to complex environments while remaining relatively straightforward to implement. Its core sampling–evaluation–update loop allows it to function as a "black-box" optimizer, simplifying implementation compared to more mathematically intensive optimization techniques.

Another key reason for choosing CEM is its flexibility with respect to action-space structure. It can naturally handle sequential decision problems by optimizing over parameterized policies or action sequences. This makes CEM a strong optimization-based benchmark that is conceptually comparable to the proposed DRL approach, while remaining independent of any learning-based mechanisms. Unlike many classical mathematical optimization algorithms, CEM can be scaled easily across both low- and high-dimensional action spaces. Its algorithmic structure does not require modification when moving to higher-dimensional problems; only a few hyperparameter adjustments, such as increasing the number of samples per iteration, are needed.

While its stochastic, sampling-based nature does not guarantee convergence to the global optimum, in reduced-dimensional environments CEM can be run with sufficiently large populations and iterations to produce high-quality solutions that are empirically close to optimal. This makes it a suitable reference method for validating our model. Moreover, even in higher-dimensional settings where reaching the true global optimum becomes unlikely, CEM remains effective at identifying relatively good solutions. Its iterative sampling and elite-selection process guides future samples toward promising regions, consistently producing solutions in areas of relatively high quality.

This characteristic makes CEM particularly valuable in the context of this thesis, as it allows for a clear empirical demonstration of the curse of dimensionality. By applying the same CEM-based optimization framework across environments of increasing size and comparing its performance to that of the DRL agent, it becomes possible to highlight both the strengths and the limitations of traditional optimization approaches and to justify the use of DRL as a scalable alternative for large-scale infrastructure maintenance problems.

5.2.2 Method overview

The CEM operates as a black-box optimizer. It treats the search for an optimal trajectory not as a descent along a gradient, but as a statistical estimation problem: identifying the sampling distribution that makes the occurrence of high-reward trajectories a "frequent" event rather than a rare one.

CEM introduces a parametric distribution $p_\theta(a_{0:T-1})$ over action sequences, where $p_\theta(a_{0:T-1}) = \Pr(\text{sample action sequence } a_{0:T-1} \mid \theta)$, and aims to find parameters θ^* that maximize the expected return, as defined in Equation (35).

$$\theta^* = \arg\max_\theta V^\pi(s) \tag{35}$$

An alternative perspective is to approach the objective by minimizing the cross-entropy, or equivalently the KL divergence, between the sampling distribution p_θ and an ideal distribution q^* concentrating all probability mass on optimal trajectories, see Equation (36)

$$\theta^* = \arg\min_\theta D_{\mathrm{KL}}(q^* \,\|\, p_\theta) \tag{36}$$

Since q^* is unknown, it is approximated with an empirical distribution over an elite set ϵ consisting of the top-ρ fraction of sampled trajectories. The parameter update then follows from maximum likelihood estimation, as given in Equation (37).

$$\theta_{k+1} = \arg\max_\theta \sum_{\tau \in \epsilon} \log p_\theta(a_{0:T-1}) \tag{37}$$

Iterating the update described in Equation (37) causes the distribution p_θ to concentrate around high-return action sequences, yielding an effective, gradient-free planning method for MDPs.

5.2.3 CEM Algorithm

The CEM is a population-based stochastic optimization algorithm. Starting from a specific state, it iteratively refines a probability distribution over action sequences to concentrate mass on high-return trajectories.

In this section, we describe how the CEM is applied to MDPs with discrete action sequences. We begin by presenting the general CEM framework as described in the literature [44]. Next, we explain the modifications we have introduced to improve the algorithm's performance and robustness. Finally, we provide the overall algorithm as used in this work.

General CEM Framework

The CEM operates over discrete action sequences with the goal of concentrating probability mass on high-return trajectories. In the context of MDPs, the algorithm requires the following inputs:

- Planning horizon T;
- Action space $\mathcal{A} = \{1, \ldots, K\}$;
- Number of samples per iteration N;
- Fraction of samples selected as elites $\rho \in (0, 1)$, yielding elite set size $K_{\text{elite}} = \lceil \rho N \rceil$;
- Smoothing parameter $\alpha \in (0, 1)$;
- Number of iterations n_{iters} (if used as the stopping criterion).

I. Initialization

The distribution over action sequences $a_{0:T-1} = (a_0, \ldots, a_{T-1})$ is modeled as a time-factorized categorical distribution, as given in Equation (38).

$$p_\theta(a_{0:T-1}) = \prod_{t=0}^{T-1} \prod_{k=1}^{K} v_{t,k}^{\mathbb{I}(a_t=k)} \tag{38}$$

where $v_{t,k} \in [0, 1]$ satisfies $\sum_{k=1}^{K} v_{t,k} = 1$ for each time step t. Initially, the distribution is uniform, reflecting equal probability for all actions, i.e., $v_{t,k} = \frac{1}{K}$ for all t and k.

II. Iterative Procedure

Each iteration of the algorithm performs the following steps:

1. Sampling and Evaluation. A set of N sequences is drawn from the distribution p_θ, and their returns $R^{(i)}$ are evaluated. The sequences are then sorted according to their returns.

2. Elite Selection. The elite set $\mathcal{E}$ is formed by taking the top K_{elite} sequences based on return. This selection is formalized in Equation (39).

$$\mathcal{E} = \{i \mid R^{(i)} \text{ is among the top } K_{\text{elite}}\} \tag{39}$$

These sequences represent the most promising candidates under the current distribution.

3. Empirical Frequency Estimation. For each action k at time t, the empirical frequency among the elite sequences is computed as described in Equation (40).

$$\hat{v}_{t,k} = \frac{1}{K_{\text{elite}}} \sum_{i \in \mathcal{E}} \mathbb{I}(a_t^{(i)} = k) \tag{40}$$

where $\mathbb{I}(\cdot)$ is the indicator function, which simply checks whether a condition is true. In this case, $\mathbb{I}(a_t^{(i)} = k)$ is 1 if the action $a_t^{(i)}$ of sequence i at time t is equal to k, and 0 otherwise. By summing this over all elite sequences and dividing by the number of elites K_{elite}, we get the fraction of elite sequences, $\hat{v}_{t,k}$, that chose action k at time t.

4. Smoothed Distribution Update. To preserve exploration and avoid premature convergence, the distribution parameters are updated using exponential smoothing, as in Equation (41).

$$v_{t,k}^{\text{new}} = \alpha \hat{v}_{t,k} + (1 - \alpha) v_{t,k}^{\text{old}} \tag{41}$$

Where α is the smoothing factor. Without smoothing, actions absent from the elite set would be permanently assigned zero probability, eliminating them from future samples.

III. Termination

Iterations continue until a stopping criterion is satisfied, which may include reaching n_{iters}, convergence of $v_{t,k}$, or stagnation in elite returns. Upon termination, the optimized distribution p_θ can be used to select high-return action sequences.

Combining these steps yields the baseline CEM algorithm summarized in Algorithm 7.

Algorithm 7 Standard Cross Entropy Method for Planning in MDPs

Require: T, K, N, ρ, α, n_{iters}
1: $K_{\text{elite}} \leftarrow N \cdot \rho$
2: Initialize $v_{t,k} = \frac{1}{K}$ for all t, k
3: $j \leftarrow 1$
4: **for** $j = 1, \ldots, n_{\text{iters}}$ **do**
5: Sample N sequences i from p_θ
6: Compute return $R(i)$ for every sequence
7: Sort sequences by $R(i)$
8: Select elite set $\mathcal{E} = \{i \mid R(i)$ among top $K_{\text{elite}}\}$
9: Frequency estimate:

$$\hat{v}_{t,k} \leftarrow \frac{1}{K_{\text{elite}}} \sum_{i \in \mathcal{E}} \mathbf{I}(a_t^i = k)$$

10: Smoothing:

$$v_{t,k}^{\text{new}} \leftarrow \alpha \hat{v}_{t,k} + (1 - \alpha) v_{t,k}^{\text{old}}$$

11: $v_{t,k}^{\text{old}} \leftarrow v_{t,k}^{\text{new}}$
12: $j \leftarrow j + 1$
13: **end for**
14: **return** $p_\theta(a_{0:T-1}) = \prod_{t=0}^{T-1} \prod_{k=1}^{K} v_{t,k}^{\mathbf{1}(a_t = k)}$

Algorithm Enhancements

In addition to the standard CEM described above, several modifications are introduced to improve robustness, encourage exploration, and accelerate convergence.

I. KL Divergence Cap (Trust-Region Updates)

KL divergence measures how much one probability distribution differs from another. The divergence between p and q is written in Equation (42), while the cap on divergence between consecutive action-sequence distributions is shown in Equation (43).

$$D_{\mathrm{KL}}(p \,\|\, q) = \sum_i p_i \log \frac{p_i}{q_i} \tag{42}$$

$$D_{\mathrm{KL}}(p_\theta^{\mathrm{new}} \,\|\, p_\theta^{\mathrm{old}}) \leq \mathrm{kl\_cap} \tag{43}$$

Using Equation (43), we can impose a cap on the KL divergence between the new and old action–sequence distributions [46]. If an unconstrained CEM update would exceed this cap, we "clip" or project the updated distribution to satisfy the bound. This ensures that each iteration's new action sequences do not vary too much, improving stability and preventing convergence to poorly optimized local maxima.

II. Minimum-Probability Floor (Always-Explore Policy)

To maintain exploration, each action is assigned a lower bound $p_{\min}$. First, the probabilities are adjusted to enforce this minimum, as shown in Equation (44). Then they are renormalized to ensure the probabilities sum to one (Equation (45)).

$$\mathrm{new\_prob}[k] = \max(\mathrm{new\_prob}[k], p_{\min}) \tag{44}$$

$$\mathrm{new\_prob} \leftarrow \frac{\mathrm{new\_prob}}{\sum_k \mathrm{new\_prob}[k]} \tag{45}$$

This guarantees that every action retains a minimal probability while maintaining a valid probability distribution. Preventing the search distribution from collapsing and preserving policy improvement.

III. Weighted Action Counting (Reward-Weighted Updates)

Instead of treating each elite trajectory equally (as in Equation (40)), we weight each elite sequence by its return when updating action probabilities. For action k at time t, the weighted count is computed as shown in Equation (46).

$$\hat{v}_{t,k} = \frac{1}{K_{\mathrm{elite}}} \sum_{i \in \mathcal{E}} w_i \, \mathbb{I}(a_t^{(i)} = k) \tag{46}$$

Where $\mathbb{I}(\cdot)$ is the indicator function and w_i is a weight proportional to the return R_i of the elite sequence i (computed using Equation (47)).

$$w_i = \begin{cases} 1 & \text{if IQR} < \epsilon \\ \dfrac{\exp\left(\frac{R_i - m}{\text{IQR}}\right)}{\sum_j \exp\left(\frac{R_j - m}{\text{IQR}}\right)} & \text{otherwise} \end{cases} \tag{47}$$

where

- R_i: Return of the i-th elite sequence;
- $m = \text{median}\{R_1, \ldots, R_{N_e}\}$;
- $\text{IQR} = Q_{75}(R) - Q_{25}(R)$ (interquartile range of returns).

Together, Equations (46) and (47) bias the search toward the most promising trajectories, speeding convergence while retaining the basic CEM logic.

IV. Dynamic Parameter Scheduling (Annealing Curriculum)

While standard CEM uses fixed hyperparameters, we can divide training into phases with gradually decreasing exploration. Early iterations employ "wide" settings to encourage broad exploration (e.g., high entropy or large sampling variance), and the parameters are progressively reduced to focus the search. This approach mirrors simulated annealing: a high "temperature" at the start allows the algorithm to explore the solution space broadly, while a cooling schedule gradually concentrates the search on the most promising regions [47].

In practice, we implement a three-phase schedule (exploration $\rightarrow$ balanced $\rightarrow$ exploitation), annealing our hyperparameters accordingly. Over time, we reduce the number of samples per iteration, the KL cap, and $p_{\min}$. This curriculum-style schedule ensures that the algorithm first samples diverse trajectories and then refines around high-reward paths, improving both robustness and final solution quality.

Accounting for all of these enhancements, the resulting algorithm is summarized in Algorithm 8.

Algorithm 8 Cross Entropy Method for Planning in MDPs (Enhanced)

Require: T, K, N, ρ, α, n_{iters}, $n_{\text{iters}}^{\text{explor}}$, $KL_{\text{cap}}^{\text{explor}}$, $p_{\min}^{\text{explor}}$, N_{explor}, $n_{\text{iters}}^{\text{bal}}$, $KL_{\text{cap}}^{\text{bal}}$, $p_{\min}^{\text{bal}}$, N_{bal}, $KL_{\text{cap}}^{\text{exploit}}$, $p_{\min}^{\text{exploit}}$, N_{exploit}

$K_{\text{elite}} \leftarrow N \cdot \rho$
Initialize $v_{t,k} = \frac{1}{K}$ for all t, k
$j \leftarrow 1$
for $j = 1, \ldots, n_{\text{iters}}$ **do**
  if $j \leq n_{\text{iters}}^{\text{explor}}$ **then** # Exploration phase
    $N \leftarrow N_{\text{explor}}$
    $KL_{\text{cap}} \leftarrow KL_{\text{cap}}^{\text{explor}}$
    $p_{\min} \leftarrow p_{\min}^{\text{explor}}$
  else if $j \leq n_{\text{iters}}^{\text{bal}}$ **then** # Balance phase
    $N \leftarrow N_{\text{bal}}$
    $KL_{\text{cap}} \leftarrow KL_{\text{cap}}^{\text{bal}}$
    $p_{\min} \leftarrow p_{\min}^{\text{bal}}$
  else # Exploitation phase
    $N \leftarrow N_{\text{exploit}}$
    $KL_{\text{cap}} \leftarrow KL_{\text{cap}}^{\text{exploit}}$
    $p_{\min} \leftarrow p_{\min}^{\text{exploit}}$
  end if
  Sample N sequences from p_θ
  Compute return $R(i)$ for for every sequence
  Sort sequences by $R(i)$
  Select elite set $\mathcal{E} = \{i \mid R(i) \text{ among top } K_{\text{elite}}\}$
  Frequency estimate:

$$\hat{v}_{t,k} \leftarrow \frac{1}{K_{\text{elite}}} \sum_{i \in \mathcal{E}} w_i \, \mathbf{I}(a_t^{(i)} = k), \quad w_i = \begin{cases} 1, & \text{if IQR} < \epsilon \\ \dfrac{\exp\left(\dfrac{R_i - m}{\text{IQR}}\right)}{\sum_j \exp\left(\dfrac{R_j - m}{\text{IQR}}\right)} & \text{otherwise} \end{cases}$$

  Smoothing:

$$v_{t,k}^{\text{new}} \leftarrow \alpha \hat{v}_{t,k} + (1 - \alpha) v_{t,k}^{\text{old}}$$

  if $D_{\text{KL}}\left(v_{t,k}^{\text{new}} \,\|\, v_{t,k}^{\text{old}}\right) \leq KL_{\text{cap}}$ **then**
    $v_{t,k}^{\text{new}} \leftarrow v_{t,k}^{\text{new}}$
  else
    $\beta = \arg\max_{\beta \in [0,1]} \; D_{\text{KL}}\left(v_{t,k}^{\text{new}}(\beta) \,\|\, v_{t,k}^{\text{old}}\right) \leq KL_{\text{cap}}$
    $v_{t,k}^{\text{new}} \leftarrow (1 - \beta) v_{t,k}^{\text{old}} + \beta v_{t,k}^{\text{new}}$
  end if
  $v_{t,k}^{\text{new}} \leftarrow \max(v_{t,k}^{\text{new}}, \; p_{\min})$
  Renormalize: $v_{t,k}^{\text{new}} \leftarrow \frac{v_{t,k}^{\text{new}}}{\sum_{k'} v_{t,k'}^{\text{new}}}, \quad \forall t$
  $v_{t,k}^{\text{old}} \leftarrow v_{t,k}^{\text{new}}$
  $j \leftarrow j + 1$
end for
return $p_\theta(a_{0:T-1}) = \prod_{t=0}^{T-1} \prod_{k=1}^{K} v_{t,k}^{\mathbf{1}(a_t = k)}$

6 Methodology III: Constraint-Aware DRL Framework

This chapter builds on the unconstrained DRL framework developed in Chapter 4, in which maintenance strategies were optimized solely with respect to total life-cycle costs. While this formulation is effective for establishing a theoretical baseline and validating the learning architecture, it abstracts away from the operational realities of public infrastructure management.

In real-world settings, optimal maintenance planning is not defined by cost minimization alone. Decisions must comply with safety requirements, respect fixed budgetary cycles, and limit the societal disruption caused by simultaneous interventions. These considerations impose explicit restrictions on the feasible solution space and fundamentally alter the nature of the optimization problem. A DRL agent that ignores such constraints may converge to policies that are efficient in theory but infeasible or unacceptable in practice.

The objective of this chapter is therefore to extend the DRL framework to a constraint-aware formulation. Constraints are incorporated directly into the learning process using a combination of complementary mechanisms: hard constraints that are enforced at the decision level and soft, long-term constraints that are handled through penalty-based optimization. This allows the agent to learn policies that are not only cost-effective but also operationally feasible.

The remainder of this chapter is structured to guide the reader from problem formulation to algorithmic implementation and optimization. We begin in Section 6.1 by formally defining the operational constraints and their mathematical bounds. In Section 6.2, we introduce the necessary state augmentations required to make these constraints observable to the agent. Subsequently, Sections 6.3 and 6.4 detail the implementation of action masking for enforcing hard constraints, while Sections 6.5-6.8 present the PID-Lagrangian relaxation method for handling global, soft constraints. Finally, Section 6.9 proposes a curriculum learning strategy to accelerate training in these complex, constrained environments.

6.1 Constraint Definition

This section formalizes the constraints incorporated into the DRL framework. These constraints are derived from practical limitations faced by decision makers such as the Municipality of Amsterdam when planning quay wall maintenance and are introduced to ensure that the learned policies remain feasible for real-world deployment.

The constraints considered in this thesis are grouped into three categories. First, risk-related constraints limit unacceptable levels of structural degradation or failure impact, ensuring that safety requirements are satisfied throughout the planning horizon. Second, financial constraints reflect municipal budgetary practices by restricting maintenance expenditures within predefined annual limits. Third, societal disruption constraints bound the level of

urban hindrance caused by maintenance activities, accounting for the cumulative impact of road and parking closures on city functioning.

In this section, each constraint category is described conceptually and subsequently expressed as a mathematical condition within the MDP formulation. This formalization provides the basis for the constraint-handling mechanisms introduced in the subsequent sections.

6.1.1 Constraints on Failure and Risk

From a municipal perspective, it is essential to ensure that maintenance policies do not lead to unacceptable safety risks or levels of structural degradation, even if such policies might reduce costs. A purely cost-driven policy could, for example, defer interventions indefinitely, resulting in highly deteriorated infrastructure with elevated failure risks. To prevent such outcomes, constraints related to structural failure are introduced.

Two complementary approaches are considered: constraining the probability of failure at the component level, and constraining the expected failure-related costs at the network level.

I. Maximum Probability of Failure per Component

The first failure-related constraint limits the maximum allowable probability that any individual quay wall segment transitions into the failed state. This constraint enables the municipality to explicitly control how risky maintenance policies are allowed to be at the component level, thereby biasing decisions toward more conservative or more risk-tolerant strategies as desired.

The mathematical modelling for this constraint is defined in Equation (48):

$$\begin{aligned} \pi^* = \arg\max_{\pi} \quad & \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right] \\ \text{subject to:} \quad & P_{\text{fail}}(s_i^t, a_i^t) \leq P_{\text{fail}}^{\max}, \quad \forall i = 1, \ldots, N, \quad \forall t = 0, \ldots, T \end{aligned} \tag{48}$$

This constraint enforces an upper bound on the failure probability of each component at every decision epoch.

II. Maximum Failure Costs

Rather than constraining failure risk directly, an alternative approach is to limit the expected societal cost associated with failures. From a policy perspective, this reflects a situation in which the municipality allocates a fixed annual budget to absorb the societal consequences of quay wall failures, such as safety risks, emergency responses, and economic disruption.

Unlike failure probabilities, failure costs are aggregated across the entire network, meaning that decisions made for one component affect the remaining budget available for others. As a result, this constraint introduces coupling between components at the system level.

The mathematical formulation is provided in Equation (49):

$$\begin{aligned} \pi^* = \arg\max_{\pi} \quad & \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right] \\ \text{subject to:} \quad & C^t_{\text{failure}} = \sum_{i=1}^{N} P_{\text{fail}}(s_i^t, a_i^t)\, c_i^{\text{failure}} \leq C^{\max}_{\text{failure}}, \quad \forall t = 0, \ldots, T \end{aligned} \tag{49}$$

This formulation ensures that the total expected failure-related cost incurred across all quay wall segments in a given year does not exceed a predefined threshold.

6.1.2 Constraints on Maintenance Budget

Another critical class of constraints concerns the financial resources available for maintenance interventions. Municipal budgets are typically allocated according to fixed accounting rules, which can vary depending on political priorities and long-term planning strategies. Incorporating budget constraints ensures that the DRL agent produces policies that are not only cost-effective but also financially feasible.

Three budget formulations are considered in this study: a strict yearly budget, a yearly budget with budget rollover, and a total budget over the full planning horizon.

I. Yearly Budget (No Transfer)

In the simplest formulation, the municipality is allocated a fixed maintenance budget each year, and any unused funds cannot be carried over to subsequent years. This reflects conservative public-sector accounting practices. The optimization problem is defined in Equation (50):

$$\begin{aligned} \pi^* = \arg\max_{\pi} \quad & \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right] \\ \text{subject to:} \quad & C^t_{\text{direct}} = \sum_{i=1}^{N} c_i^{\text{direct}}(a_i^t) \leq B_t^{\text{year}}, \quad \forall t = 0, \ldots, T \end{aligned} \tag{50}$$

II. Yearly Budget with Money Transfer

A more flexible formulation allows unused budget from one year to be transferred to the next. This reflects planning practices in which delayed interventions can be compensated by higher spending in later years. The corresponding formulation is shown in Equation (51):

$$\begin{aligned} \pi^* = \arg\max_{\pi} \quad & \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right] \\ \text{subject to:} \quad & C^t_{\text{direct}} = \sum_{i=1}^{N} c_i^{\text{direct}}(a_i^t) \leq B_t^{\text{avail}}, \quad \forall t = 0, \ldots, T, \\ \text{where:} \quad & B_t^{\text{avail}} = B_t^{\text{year}} + B_{t-1}^{\text{unused}}, \\ & B_{t-1}^{\text{unused}} = B_{t-1}^{\text{avail}} - \sum_{i=1}^{N} c_i^{\text{direct}}(a_i^{t-1}). \end{aligned} \tag{51}$$

III. Budget over the Full Planning Horizon

Finally, the municipality may allocate a single total budget to be spent over the entire planning horizon, allowing complete flexibility in the timing of expenditures. This is modelled as shown in Equation (52):

$$\begin{aligned} \pi^* = \arg\max_{\pi} \quad & \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right] \\ \text{subject to:} \quad & \sum_{t=0}^{T} C^t_{\text{direct}} = \sum_{t=0}^{T} \sum_{i=1}^{N} c_i^{\text{direct}}(a_i^t) \leq B_{\text{total}} \end{aligned} \tag{52}$$

6.1.3 Constraints on City Disruption

The final class of constraints addresses urban disruption caused by maintenance interventions. Unlike direct and failure costs, city costs are not additive over individual components. Instead, they depend on the joint configuration of actions across quay wall units, reflecting non-linear network effects such as traffic rerouting and congestion. This makes city disruption constraints fundamentally different and more challenging to enforce within a multi-agent DRL framework.

I. Maximum City Costs

In this constraint, the municipality specifies a maximum acceptable level of annual city disruption cost, representing societal tolerance for traffic delays, environmental impacts, and reduced accessibility. The mathematical constraint is defined in Equation (53):

$$\begin{aligned} \pi^* = \arg\max_{\pi} \quad & \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right] \\ \text{subject to:} \quad & C^t_{\text{interaction}} = \sum_{k \in \{\text{parking,traffic}\}} C^{\text{group}}_k(G^t_k) \leq C^{\max}_{\text{interaction}}, \quad \forall t = 0, \ldots, T \end{aligned} \tag{53}$$

This constraint limits the total network-level disruption caused by simultaneous maintenance actions in any given year.

6.2 Budget-Constrained State Augmentation

In Chapter 4, we introduced *Temporal State Augmentation* to address the non-stationarity of the finite planning horizon. By including normalized time in the state vector, we restored the Markov property with respect to the finite horizon, enabling the agent to distinguish between early-stage and end-of-horizon decisions.

However, the introduction of the cumulative constraints defined in Section 6.1 reintroduces a violation of the Markov property. For example, if a total budget (e.g., 10 million €) is available over 20 years, the optimal decision at any given step depends not only on the structural state and the current time, but critically on the amount of budget remaining for each year.

If the remaining budget is not included in the state, the MDPs no longer satisfies the Markov property. In that case, if the reward is penalized when the constraint is violated, identical state–action pairs may yield very different rewards, depending on whether the budget has already been partially or fully exhausted. For example, performing the same maintenance action in the same structural state may be acceptable early in the planning horizon but incur a large penalty later if the remaining budget is insufficient. From the agent's perspective, these situations are indistinguishable when the remaining budget is not part of the state.

This ambiguity significantly hinders learning. Since PPO relies on batches of transitions that are shuffled during training, the agent cannot infer whether a penalty is caused by the action itself or by an unobserved constraint violation. As a result, the policy receives conflicting learning signals for the same observed state, making it difficult to converge to an optimal solution.

To restore the Markov property, the state is augmented with the remaining budget:

$$\tilde{s}_t = (s_t, b^{\text{rem}}_t)$$

where b^{rem}_t denotes the remaining budget at decision step t. By explicitly including this

information, the state once again contains all variables necessary to predict future rewards and transitions, enabling the agent to learn constraint-aware and temporally consistent policies.

6.2.1 Normalization for Stability

Practically, as with temporal state augmentation, it is essential to ensure that all components of the augmented state are of comparable magnitude to facilitate stable and efficient learning. This consideration is particularly important when augmenting the state with the remaining budget. In the quay wall maintenance environment, budget values are expressed in absolute monetary terms and may be on the order of several millions of euros, whereas the other state variables typically take values in a much smaller range (between 0 and 1). Without normalization, this large discrepancy in scale can dominate the state representation and adversely affect the learning dynamics of the policy network, leading to poor convergence or unstable training behavior.

To address this issue, the remaining budget is normalized by the initial total budget available over the planning horizon. The normalized budget variable is therefore defined as in Equation (54).

$$b_t^{\text{norm}} = \frac{b_t^{\text{rem}}}{b^{\text{init}}} \tag{54}$$

where:

- b_t^{norm} denotes the normalized remaining budget at decision step t,
- b_t^{rem} denotes the remaining absolute budget at decision step t, and
- b^{init} denotes the initial total budget available over the planning horizon.

This yields a dimensionless quantity in the range $[0, 1]$. By expressing the remaining budget as a fraction of the initial budget, the agent learns to condition its decisions on the relative availability of financial resources rather than on absolute monetary values.

6.3 Action Masking

With the operational constraints mathematically formalized and the state space augmented to track resource availability, the next step is to enforce these boundaries within the learning process. Specifically, we address the implementation of "hard" constraints—those that represent strictly inviolable limits at any given decision step. In the context of quay wall maintenance, these constraints correspond to actions that are operationally or financially

prohibited, such as attempting a costly renovation when the annual budget is already exhausted, or neglecting a critical component that has exceeded its maximum allowable failure probability.

To strictly enforce these limitations within the PPO framework, we utilize action masking. Unlike soft penalty methods that merely discourage violations through negative rewards, action masking dynamically filters the agent's action space based on the current state and remaining resources. This ensures that invalid actions are mathematically excluded from the policy distribution, guaranteeing that the agent effectively never selects an action that violates a hard constraint.

In the broader RL literature, handling state-dependent action sets is a well-established challenge. Sampling invalid actions can result in wasted environment interactions, unstable training dynamics, or undefined environment transitions. Consequently, action masking was originally introduced to prevent agents from repeatedly selecting invalid moves in environments with large discrete action spaces, such as real-time strategy games [48, 49, 50].

More recently, action masking has been explicitly integrated into policy-gradient methods. In particular, Tang et al. [51] proposed incorporating an action mask into the PPO algorithm, showing that masking invalid actions can significantly improve learning efficiency and final performance when state-dependent action feasibility constraints are present.

This section formalizes the action masking mechanism and explains how it interacts with the policy representation and policy gradient computation in PPO.

6.3.1 Policy Representation in PPO

As discussed in Chapter 4, PPO is an actor–critic method in which the policy (actor) is parameterized by a neural network with parameters θ. Given a state s_t, the actor network outputs a vector of real-valued logits:

$$l_\theta(s_t) = [l_\theta(a_0 \mid s_t), \ldots, l_\theta(a_N \mid s_t)],$$

with one logit corresponding to each action in the discrete action space.

$$\mathcal{A} = \{a_0, a_1, a_2, \ldots, a_N\}$$

These logits are transformed into a probability distribution over actions via the softmax function (Equation (55)):

$$\pi_\theta(a_i \mid s_t) = \frac{\exp(l_\theta(a_i \mid s_t))}{\sum_{j=0}^{N} \exp(l_\theta(a_j \mid s_t))} \tag{55}$$

By construction, the policy satisfies $0 < \pi_\theta(a_i \mid s_t) < 1$ and $\sum_{i=0}^{N} \pi_\theta(a_i \mid s_t) = 1$. At each timestep t, the agent samples an action according to $a_t \sim \pi_\theta(\cdot \mid s_t)$.

6.3.2 Action Space Example

In the considered case study, the discrete action space consists of five actions:

$$\mathcal{A} = \{a_0, a_1, a_2, a_3, a_4\},$$

where a_0 corresponds to "do nothing" and a_4 corresponds to "full renovation." At the beginning of training, the policy network is typically initialized such that the action probabilities are uniform:

$$\pi_\theta(\cdot \mid s_0) = [0.2, 0.2, 0.2, 0.2, 0.2].$$

As training progresses, the parameters θ are updated and the policy becomes state-dependent and non-uniform.

6.3.3 State-Dependent Invalid Actions

In many environments, action feasibility depends on the current state. Let $\mathcal{A}_{\text{valid}}(s_t) \subseteq \mathcal{A}$ denote the subset of valid actions in state s_t. Suppose that in state s_0, action a_1 is invalid, such that:

$$\mathcal{A}_{\text{valid}}(s_0) = \{a_0, a_2, a_3, a_4\}.$$

Without additional constraints, PPO would still assign a non-zero probability to a_1. The agent would therefore need to learn indirectly, through repeated penalties or corrections, to reduce the probability of selecting this invalid action.

6.3.4 Action Masking at the Logit Level

Action masking prevents this behavior by modifying the logits before the softmax operation. Define a binary action mask (Equation (56)):

$$m(a_i \mid s_t) = \begin{cases} 1, & \text{if } a_i \in \mathcal{A}_{\text{valid}}(s_t) \\ 0, & \text{otherwise.} \end{cases} \tag{56}$$

The masked logits are defined as stated in Equation (57):

$$\tilde{l}_\theta(a_i \mid s_t) = \begin{cases} l_\theta(a_i \mid s_t), & \text{if } m(a_i \mid s_t) = 1 \\ -\infty, & \text{if } m(a_i \mid s_t) = 0 \end{cases} \tag{57}$$

where in practice $-\infty$ is approximated by a very large negative constant (e.g., -3.4×10^{38}). The resulting masked policy is then given by Equation (58):

$$\pi_\theta^m(a_i \mid s_t) = \frac{\exp(\tilde{l}_\theta(a_i \mid s_t))}{\sum_{j=0}^{N} \exp(\tilde{l}_\theta(a_j \mid s_t))} \tag{58}$$

For the example above, this yields:

$$\pi_\theta^m(\cdot \mid s_0) = [0.25, 0.0, 0.25, 0.25, 0.25].$$

Thus, invalid actions are assigned exactly zero probability, while the probabilities of valid actions are renormalized over $\mathcal{A}_{\text{valid}}(s_t)$. Figure 11 provides a visual illustration of this masking operation at the logit level.

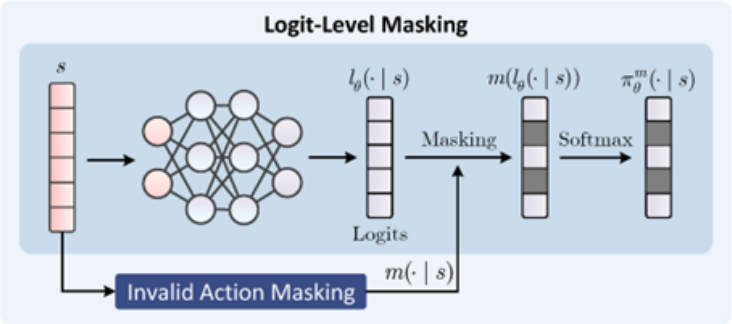

Figure 11: Visualisation of an action mask at the logit level [4].

6.3.5 Effect on Policy Gradient Updates

Action masking has an important consequence beyond probability renormalization. Because the probability of an invalid action is identically zero, the corresponding log-probability satisfies $\log \pi_\theta^m(a_i \mid s_t) = -\infty$, which is constant with respect to θ. Consequently:

$$\nabla_\theta \log \pi_\theta^m(a_i \mid s_t) = 0.$$

As a result, invalid actions do not contribute to the policy gradient. For a trajectory $\tau = (s_0, a_0, \ldots, s_{T-1}, a_{T-1})$, the masked policy gradient can be written as:

$$g_{\text{masked}} = \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta^m(a_t \mid s_t)\, G_t \right],$$

where G_t denotes the return or advantage estimate at timestep t. This formulation ensures that learning updates are restricted exclusively to valid actions, effectively enforcing hard action feasibility constraints directly at the policy level.

6.3.6 Summary

This way, action masking incorporates state-dependent feasibility constraints into PPO by modifying the policy's logits prior to the softmax operation. This guarantees that invalid actions are never sampled and that their corresponding policy gradients are identically zero. As demonstrated in prior work [51], this simple yet effective modification can substantially improve learning efficiency and stability in constrained decision-making problems.

6.4 Constraint Enforcement via Action Masking in PPO

As shown by Tang et al. [51], invalid action masking applied at the policy level is fully compatible with PPO. Since the policy update is still performed using the standard clipped surrogate objective, action masking does not modify the underlying optimization problem, but instead restricts the policy's support to a dynamically feasible subset of the action space. This makes action masking a practical and principled mechanism for enforcing hard constraints in reinforcement learning.

Despite this theoretical compatibility, implementing action masking on top of PPO is non-trivial and highly dependent on the structure of the environment, the coupling between decision variables, and whether constraints are local (component-wise) or global (network-wide).

In this thesis, action masking is used as the primary mechanism to enforce the constraints introduced in Section 6.1. Each constraint induces a state-dependent feasible action set, and the exact implementation of the masking logic differs across constraint types. This section describes how action masking is integrated into PPO for the quay wall maintenance case study and how each constraint is translated into a mathematically well-defined masking rule.

Time augmentation, to preserve the Markov property, is used in all experiments. For several constraints discussed below, additional state augmentation is required so that the agent has explicit access to resource availability (e.g., remaining budget). These architectural choices are explained alongside each constraint.

Throughout this section, the standard PPO algorithm (Algorithms 4, 5, and 6 from Chapter 4) is treated as the base. Constraint-specific variants are derived from it, and deviations from standard PPO are highlighted in blue in the corresponding algorithms. The rollout and learning phases are discussed separately when needed.

6.4.1 Constraints on Failure

I. Maximum Probability of Failure

Constraint Integration

As defined in Section 6.1, the Maximum Probability of Failure constraint is formulated by Equation (59).

$$P_{\text{fail}}(s_i^t, a_i^t) \leq P_{\text{fail}}^{\max}, \quad \forall i = 1, \ldots, N, \quad \forall t \tag{59}$$

To apply action masking, at each decision step, actions that violate this inequality are considered infeasible and must be excluded from the policy's support. Formally, for each component and state , a binary mask is constructed as shown in Equation (60):

$$m_i(a \mid s_t) = \begin{cases} 1, & \text{if } P_{\text{fail}}(s_i^t, a_i^t) \leq P_{\text{fail}}^{\max} \\ 0, & \text{otherwise} \end{cases} \tag{60}$$

Infeasible actions (i.e., actions with $m_i(a \mid s_t) = 0$) receive zero probability mass, while the remaining actions are renormalized.

State Representation

No additional state augmentation is required for this constraint. The state vector is defined as:

$$s_t = [\tilde{s}_t, t/T]$$

Implementation in PPO

Since the feasibility of actions depends only on the current component state, masking can be recomputed independently for each component. The same masking logic is applied during the rollout phase, when actions are sampled to ensure only feasible actions are selected, and during the learning phase, where the same mask is used to compute log-probabilities and PPO probability ratios. A detailed explanation of the rollout implementation is provided in Algorithm 9, and the learning-phase implementation is shown in Algorithm 10 and Algorithm 11.

Algorithm 9 PPO Rollout with Failure-Probability Masking

Require: Hyperparameters

Blue: Deviations from Standard PPO

max_failure_probability ← Load maximum allowed failure probability
Initialize policy (actor) network weights θ
Initialize value function (critic) network weights ϕ
for $i = 1, \ldots,$ train_iters **do**
  $t \leftarrow 0$, $s_t \leftarrow$ reset environment
  for $j = 1, \ldots,$ steps_per_iter **do**
    $t \leftarrow t + 1$
    $\ell_\theta(\cdot \mid s_t)$, $\pi_\theta(\cdot \mid s_t) \leftarrow$ ActorNet(s_t) # Forward pass of actor network
    $V_\phi(s_t) \leftarrow$ CriticNet(s_t) # Forward pass of critic network
    Initialize $\pi_\theta^m(\cdot \mid s_t), \tilde{\ell}_\theta(\cdot \mid s_t)$
    Initialize $m(\cdot \mid s_t)$ where $m(a \mid s_t) = 1, \ \forall a \in \mathcal{A}$
    for comp ∈ total_components **do**
      if failure_probability$(a_3,$ comp$)$ > max_failure_probability **then**
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 0, 0, 0, 1]$ # Mask a_0, a_1, a_2, a_3
      else if failure_probability$(a_2,$ comp$)$ > max_failure_probability **then**
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 0, 0, 1, 1]$ # Mask a_0, a_1, a_2
      else if failure_probability$(a_1,$ comp$)$ > max_failure_probability **then**
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 0, 1, 1, 1]$ # Mask a_0, a_1
      else if failure_probability$(a_0,$ comp$)$ > max_failure_probability **then**
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 1, 1, 1, 1]$ # Mask a_0
      else
        pass # Failure probability within limit: no masking
      end if
      $\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}} = \begin{cases} \ell_\theta(a_i \mid s_t)_{\text{comp}}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 1 \\ -3.4 \times 10^{38}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 0 \end{cases}$
      $\pi_\theta^m(a_i \mid s_t)_{\text{comp}} = \frac{\exp(\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}})}{\sum_{j=0}^{N} \exp(\tilde{\ell}_\theta(a_j \mid s_t)_{\text{comp}})}$
      $a_{t,\text{comp}} \leftarrow$ sample from $\pi_\theta^m(\cdot \mid s_t)_{\text{comp}}$
    end for
    $\pi_\theta^m(a_t \mid s_t) \leftarrow$ evaluate probability of a_t under $\pi_\theta^m(\cdot \mid s_t)$
    $s_{t+1}, R(s_t, a_t) \leftarrow$ environment step

Where: $s_t = [\tilde{s}_t, \ \frac{t}{T}] \ \Rightarrow \ s_{t+1} = [\tilde{s}_{t+1}, \ \frac{t+1}{T}]$

    Store tuple $\{s_t, a_t, \pi_\theta^m(a_t \mid s_t), V_\phi(s_t), R(s_t, a_t)\}$ in Rollout Memory D_k
    $s_t \leftarrow s_{t+1}$
    if $t = T$ **or** $j =$ steps_per_iter **then**
      if $t = T$ **then**
        $V_\phi(s_{t+1}) \leftarrow 0$
        $s_t \leftarrow$ reset environment
      else
        $V_\phi(s_{t+1}) \leftarrow$ CriticNet(s_{t+1})
      end if
      $\delta_t \leftarrow R(s_t, a_t) + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
      Compute returns: $G_t \leftarrow \sum_{l=0}^{T-t-1} \gamma^l R_{t+l}$
      Compute advantages: $A_t \leftarrow \sum_{l=0}^{T-t-1} (\gamma\tau)^l \delta_{t+l}$
      Store δ_t, A_t in D_k
    end if
  end for
  Train Agent (D_k) using Algorithm 10 # training of networks
end for

Algorithm 10 PPO Update with Failure-Probability Masking

Blue: Deviations from Standard PPO

Require: Rollout Memory $D_k \rightarrow$ shuffled and divided into minibatches (over multiple epochs)

1: **Update Critic Network:** Update parameters ϕ, using the Critic loss function:

$$L_{\text{Critic}}(\phi) = \frac{1}{T} \sum_{t=1}^{T} \left(V_\phi(s_t) - G_t \right)^2$$

2: **Update Actor Network:** Update parameters θ, using the Actor loss function with entropy regularization:

$$L_{\text{Actor}}(\theta) = \sum_{t=1}^{T} \left[\min \left(r_t(\theta) \hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_t \right) + c_2\, H[\pi_\theta^m(\cdot \mid s_t)] \right]$$

3: where:

$$r_t(\theta) = \exp \left(\log \pi_\theta^m(a_t \mid s_t) - \log \pi_{\theta_{\text{old}}}^m(a_t \mid s_t) \right) \quad \text{(computed following } \textbf{Algorithm 11)}$$

$$\hat{A}_t = \frac{A_t - \mu_A}{\sigma_A}$$

$$H[\pi_\theta^m(\cdot \mid s_t)] = -\sum_a \pi_\theta^m(a \mid s_t) \log \pi_\theta^m(a \mid s_t)$$

Algorithm 11 Probability Ratio Computation with Failure-Probability Masking

Blue: Deviations from Standard PPO

Require: Rollout Memory $D_k \rightarrow$ shuffled and divided into minibatches (over multiple epochs)
Require: Current policy parameters θ

Retrieve s_t, a_t, $\pi^m_{\theta_{\text{old}}}(a_t \mid s_t)$ from rollout memory D_k
Compute current policy logits and distribution:

$$\ell_\theta(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \leftarrow \text{ActorNet}_\theta(s_t)$$

Initialize $\pi^m_\theta(\cdot \mid s_t), \tilde{\ell}_\theta(\cdot \mid s_t)$
Initialize $m(\cdot \mid s_t)$ where $m(a \mid s_t) = 1, \ \forall a \in \mathcal{A}$
for comp $\in$ total_components **do**
    if failure_probability$(a_3, \text{comp}) >$ max_failure_probability **then**
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 0, 0, 0, 1]$ # Mask a_0, a_1, a_2, a_3
    else if failure_probability$(a_2, \text{comp}) >$ max_failure_probability **then**
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 0, 0, 1, 1]$ # Mask a_0, a_1, a_2
    else if failure_probability$(a_1, \text{comp}) >$ max_failure_probability **then**
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 0, 1, 1, 1]$ # Mask a_0, a_1
    else if failure_probability$(a_0, \text{comp}) >$ max_failure_probability **then**
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 1, 1, 1, 1]$ # Mask a_0
    else
        pass # Failure probability within limit: no masking
    end if
    $\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}} = \begin{cases} \ell_\theta(a_i \mid s_t)_{\text{comp}}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 1 \\ -3.4 \times 10^{38}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 0 \end{cases}$
    $\pi^m_\theta(a_i \mid s_t)_{\text{comp}} = \frac{\exp(\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}})}{\sum_{j=0}^{N} \exp(\tilde{\ell}_\theta(a_j \mid s_t)_{\text{comp}})}$
end for
Compute the probability of the taken action a_t under $\pi^m_\theta(a_t \mid s_t)$:

$$\pi^m_\theta(a_t \mid s_t)$$

Compute probability ratio using masked log-probabilities:

$$r_t(\theta) \leftarrow \exp\Big(\log \pi^m_\theta(a_t \mid s_t) - \log \pi^m_{\theta_{\text{old}}}(a_t \mid s_t) \Big)$$

II. Maximum Failure Costs

Constraint Integration

The Maximum Failure Costs constraint limits the total monetary impact of failures across all quay walls during a single year. Formally, this limit is expressed in Equation (61).

$$C^t_{\text{failure}} = \sum_{i=1}^{N} P_{\text{fail}}(s^t_i, a^t_i)\, c^{\text{failure}}_i \leq C^{\max}_{\text{failure}}, \quad \forall t \tag{61}$$

Action masking is applied sequentially. At each step, the remaining budget for allowable failure costs is updated based on the actions already selected. Only actions that do not exceed

the remaining budget are considered feasible; the rest are masked out. For component i in state s_t, the mask is determined by Equation (62):

$$m_i(a \mid s_t) = \begin{cases} 1, & \text{if } P_{\text{fail}}(s_t, a)\, c_i^{\text{failure}} \leq B_t^{\text{rem}} \\ 0, & \text{otherwise} \end{cases} \tag{62}$$

where B_t^{rem} denotes the remaining yearly failure budget before evaluating component i. After selecting an action for component i, the remaining budget is updated according to Equation (63):

$$B_t^{\text{rem}} \leftarrow B_t^{\text{rem}} - P_{\text{fail}}(s_t, a)\, c_i^{\text{failure}} \tag{63}$$

To avoid systematic biases caused by the evaluation order, components are shuffled randomly before applying action masking. This is critical because the feasibility of later components depends on the remaining budget after earlier actions. Without shuffling, early components would consistently have more flexibility, while later components could be overly constrained, making the learned policy order-dependent. Random shuffling ensures that the agent learns a robust, permutation-invariant strategy that generalizes across all possible sequences of component evaluation.

State Representation

No additional state augmentation is required. The state vector remains as defined earlier:

$$s_t = [\tilde{s}_t, t/T]$$

Implementation in PPO

During rollout, actions are sequentially masked based on the remaining budget, and the resulting masking matrix is stored for reuse during the learning phase. This is necessary because feasible actions for each component depend on prior decisions, and reconstructing this context during learning would otherwise be computationally expensive due to component shuffling and sequential budget consumption. Storing and reapplying the masks ensures PPO updates respect the constraints and compute gradients correctly. This approach will also be necessary for more complex scenarios, such as when remaining budget can be carried over across years. Algorithm 12 details the rollout implementation, while Algorithm 13 and Algorithm 14 describe the learning-phase procedures.

Algorithm 12 PPO Rollout with Yearly Failure-Cost Masking

Require: Hyperparameters

Blue: Deviations from Standard PPO

failure_yearly_budget ← Load maximum yearly failure costs
Initialize policy (actor) network weights θ
Initialize value function (critic) network weights ϕ
for $i = 1, \ldots,$ train_iters **do**
  $t \leftarrow 0$, $s_t \leftarrow$ reset environment
  for $j = 1, \ldots,$ steps_per_iter **do**
    $t \leftarrow t + 1$
    $\ell_\theta(\cdot \mid s_t)$, $\pi_\theta(\cdot \mid s_t) \leftarrow$ ActorNet(s_t) # Forward pass of actor network
    $V_\phi(s_t) \leftarrow$ CriticNet(s_t) # Forward pass of critic network
    Initialize $\pi_\theta^m(\cdot \mid s_t), \tilde{\ell}_\theta(\cdot \mid s_t)$
    Initialize $m(\cdot \mid s_t)$ where $m(a \mid s_t) = 1, \ \forall a \in \mathcal{A}$
    remaining_budget ← failure_yearly_budget
    Shuffle total_components # Shuffle the list of all the components
    for comp ∈ total_components **do**
      if remaining_budget ≥ failure_cost$(a_0,$ comp$)$ **then**
        pass # Failure cost does not exceed limit witout action: no masking
      else if remaining_budget ≥ failure_cost$(a_1,$ comp$)$ **then**
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 1, 1, 1, 1]$ # Mask a_0
      else if remaining_budget ≥ failure_cost$(a_2,$ comp$)$ **then**
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 0, 1, 1, 1]$ # Mask a_0, a_1
      else if remaining_budget ≥ failure_cost$(a_3,$ comp$)$ **then**
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 0, 0, 1, 1]$ # Mask a_0, a_1, a_2
      else
        $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [0, 0, 0, 0, 1]$ # Mask a_0, a_1, a_2, a_3
      end if
      $$\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}} = \begin{cases} \ell_\theta(a_i \mid s_t)_{\text{comp}}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 1 \\ -3.4 \times 10^{38}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 0 \end{cases}$$
      $$\pi_\theta^m(a_i \mid s_t)_{\text{comp}} = \frac{\exp(\tilde{\ell}_\theta(a_i|s_t)_{\text{comp}})}{\sum_{j=0}^{N} \exp(\tilde{\ell}_\theta(a_j|s_t)_{\text{comp}})}$$
      $a_{t,\text{comp}} \leftarrow$ sample from $\pi_\theta^m(\cdot \mid s_t)_{\text{comp}}$
      remaining_budget ← remaining_budget − failure_cost$(a_t,$ comp$)$
    end for
    $\pi_\theta^m(a_t \mid s_t) \leftarrow$ evaluate probability of a_t under $\pi_\theta^m(\cdot \mid s_t)$
    $s_{t+1}, R(s_t, a_t) \leftarrow$ environment step

$$\textbf{Where: } s_t = [\tilde{s}_t, \ \tfrac{t}{T}] \ \Rightarrow \ s_{t+1} = [\tilde{s}_{t+1}, \ \tfrac{t+1}{T}]$$

    Store tuple $\{s_t, a_t, \pi_\theta^m(a_t \mid s_t), V_\phi(s_t), R(s_t, a_t), m(\cdot \mid s_t)\}$ in Rollout Memory D_k
    $s_t \leftarrow s_{t+1}$
    if $t = T$ **or** $j =$ steps_per_iter **then**
      if $t = T$ **then**
        $V_\phi(s_{t+1}) \leftarrow 0$
        $s_t \leftarrow$ reset environment
      else
        $V_\phi(s_{t+1}) \leftarrow$ CriticNet(s_{t+1})
      end if
      $\delta_t \leftarrow R(s_t, a_t) + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
      Compute returns: $G_t \leftarrow \sum_{l=0}^{T-t-1} \gamma^l R_{t+l}$
      Compute advantages: $A_t \leftarrow \sum_{l=0}^{T-t-1} (\gamma\tau)^l \delta_{t+l}$
      Store δ_t, A_t in D_k
    end if
  end for
  Train Agent (D_k) using Algorithm 13 # training of networks
end for

Algorithm 13 PPO Update with Stored Masks

Blue: Deviations from Standard PPO

Require: Rollout Memory $D_k \rightarrow$ shuffled and divided into minibatches (over multiple epochs)

1: **Update Critic Network:** Update parameters ϕ, using the Critic loss function:

$$L_{\text{Critic}}(\phi) = \frac{1}{T}\sum_{t=1}^{T}\left(V_\phi(s_t) - G_t\right)^2$$

2: **Update Actor Network:** Update parameters θ, using the Actor loss function with entropy regularization:

$$L_{\text{Actor}}(\theta) = \sum_{t=1}^{T}\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right) + c_2\, H[\pi_\theta^m(\cdot \mid s_t)]\right]$$

3: where:

$$r_t(\theta) = \exp\left(\log \pi_\theta^m(a_t \mid s_t) - \log \pi_{\theta_{\text{old}}}^m(a_t \mid s_t)\right) \quad \text{(computed following } \textbf{Algorithm 14)}$$

$$\hat{A}_t = \frac{A_t - \mu_A}{\sigma_A}$$

$$H[\pi_\theta^m(\cdot \mid s_t)] = -\sum_a \pi_\theta^m(a \mid s_t)\log \pi_\theta^m(a \mid s_t)$$

Algorithm 14 Probability Ratio Computation with Stored Masks

Blue: Deviations from Standard PPO

Require: Rollout Memory $D_k \rightarrow$ shuffled and divided into minibatches (over multiple epochs)

Require: Current policy parameters θ

1: Retrieve s_t, a_t, $\pi_{\theta_{\text{old}}}^m(a_t \mid s_t)$, $m(\cdot \mid s_t)$ from rollout memory D_k

2: Compute current policy logits and distribution:

$$\ell_\theta(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \leftarrow \text{ActorNet}_\theta(s_t)$$

3: Apply action masking stored in memory:

$$\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}} = \begin{cases} \ell_\theta(a_i \mid s_t)_{\text{comp}}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 1 \\ -3.4 \times 10^{38}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 0 \end{cases}$$

$$\pi_\theta^m(a_i \mid s_t)_{\text{comp}} = \frac{\exp(\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}})}{\sum_{j=0}^{N}\exp(\tilde{\ell}_\theta(a_j \mid s_t)_{\text{comp}})}$$

4: Compute the probability of the taken action a_t under $\pi_\theta^m(a_t \mid s_t)$:

$$\pi_\theta^m(a_t \mid s_t)$$

5: Compute probability ratio using masked log-probabilities:

$$r_t(\theta) \leftarrow \exp\left(\log \pi_\theta^m(a_t \mid s_t) - \log \pi_{\theta_{\text{old}}}^m(a_t \mid s_t)\right)$$

6.4.2 Constraints on Maintenance Budgets

The following constraints govern direct maintenance expenditures rather than failure costs. Their masking logic is similar to Maximum Failure Costs, but the required state information differs to capture budget dynamics. Across all cases:

- Components are shuffled at each timestep;
- Masking vectors are stored during rollout and reused during learning;
- Learning-phase updates follow the same logic as described in Algorithm 13 and Algorithm 14.

The three algorithms presented in this section will continue to highlight deviations from standard PPO in blue, while for the last two constraints, differences relative to each other and to the first constraint will additionally be highlighted in red.

I. Yearly Budget

Constraint Integration

The Yearly Budget constraint limits total spending on maintenance actions within a single year. Let B^t_{year} denote the yearly budget at time t, and $c_{\text{direct}}(a^t_i)$ the direct cost of an action on component i. The constraint ensures that spending satisfies Equation (64):

$$\sum_{i=1}^{N} c_{\text{direct}}(a^t_i) \leq B^t_{\text{year}} \tag{64}$$

Action masking is applied by sequentially evaluating each component, with only those not exceeding the remaining budget allowed. For component i in state s_t, the mask is applied as in Equation (65):

$$m_i(a \mid s_t) = \begin{cases} 1, & \text{if } c_{\text{direct}}(a^t_i) \leq B^{\text{rem}}_t \\ 0, & \text{otherwise} \end{cases} \tag{65}$$

where B^{rem}_t denotes the remaining yearly maintenance budget before evaluating component i. After selecting an action for component i, the remaining budget is updated as shown in Equation (66):

$$B^{\text{rem}}_t \leftarrow B^{\text{rem}}_t - c_{\text{direct}}(a^t_i) \tag{66}$$

State Representation

No additional augmentation is required.

$$s_t = [\tilde{s}_t, t/T]$$

Implementation in PPO

The rollout phase applies the mask sequentially and stores the masked actions for learning. PPO updates then use the same mask. Algorithm 15 shows the rollout logic. Learning-phase updates follow the procedure described for Maximum Failure Costs (Algorithm 13 and Algorithm 14).

II. Yearly Budget with Money Transfer

Constraint Integration

This variant allows unused budget from previous years to be carried forward, as formulated in Equation (67):

$$\sum_{i=1}^{N} c_{\text{direct}}(a_i^t) \leq B_t^{\text{avail}} \tag{67}$$

where the available budget is calculated via Equation (68):

$$\begin{aligned} B_t^{\text{avail}} &= B_t^{\text{year}} + B_{t-1}^{\text{unused}} \\ B_{t-1}^{\text{unused}} &= B_{t-1}^{\text{avail}} - \sum_{i=1}^{N} c_i^{\text{direct}}(a_i^{t-1}) \end{aligned} \tag{68}$$

Masking logic remains the same, but the state must include additional variables to track cumulative spending and budget availability, using the mask defined in Equation (69):

$$m_i(a \mid s_t) = \begin{cases} 1, & \text{if } c_{\text{direct}}(a_i^t) \leq B_t^{\text{rem}} \\ 0, & \text{otherwise} \end{cases} \tag{69}$$

The budget update follows Equation (70):

$$B_t^{\text{rem}} \leftarrow B_t^{\text{rem}} - c_{\text{direct}}(a_i^t) \tag{70}$$

State Representation

$$s_t = [\tilde{s}_t, t/T, \beta_t, \gamma_t],$$

where:

- β_t encodes the normalized cumulative allocated budget, as shown in Equation (71):

$$\beta_t = \frac{(t+1) \cdot B_t^{\text{year}}}{T \cdot B_t^{\text{year}}} = \frac{t+1}{T} \tag{71}$$

- γ_t encodes the normalized accumulated money spent so far, given by Equation (72):

$$\gamma_t = \frac{\sum_{k=0}^{t} c_k^{\text{direct}}}{T \cdot B_t^{\text{year}}} \tag{72}$$

Implementation in PPO

The augmented state allows the agent to infer remaining budget and future spending opportunities. The rollout logic follows the same masking rules as the standard yearly budget, but with dynamically evolving budget availability (Algorithm 16).

III. Budget over Full Horizon

Constraint Integration

This constraint limits total spending across the entire planning horizon, as expressed in Equation (73).

$$\sum_{t=0}^{T-1} \sum_{i=1}^{N} c_{\text{direct}}(a_i^t) \leq B_{\text{total}} \tag{73}$$

Again, the masking logic is identical to the standard yearly budget, but the state must include additional variables to track accumulated spending and budget availability over time, using the mask in Equation (74):

$$m_i(a \mid s_t) = \begin{cases} 1, & \text{if } c_{\text{direct}}(a_i^t) \leq B_t^{\text{rem}} \\ 0, & \text{otherwise} \end{cases} \tag{74}$$

The remaining budget is updated according to Equation (75):

$$B_t^{\text{rem}} \leftarrow B_t^{\text{rem}} - c_{\text{direct}}(a_i^t) \tag{75}$$

State Representation

$$s_t = [\tilde{s}_t, t/T, \rho_t],$$

where ρ_t encodes the normalized remaining budget defined in Equation (76):

$$\rho_t = \frac{B_t^{\text{rem}}}{B_{\text{total}}} \tag{76}$$

Implementation in PPO

Masking follows the same component-wise logic as before, but feasibility is now tied to a single remaining budget variable that spans the full horizon. The changes are summarized in Algorithm 17.

Algorithm 15 PPO Rollout with Yearly Maintenance Budget Masking

Require: Hyperparameters

Blue: Deviations from Standard PPO

yearly_budget ← Load yearly budget
Initialize policy (actor) network weights θ
Initialize value function (critic) network weights ϕ
for $i = 1, \ldots,$ train_iters **do**
  $t \leftarrow 0$, $s_t \leftarrow$ reset environment
  for $j = 1, \ldots,$ steps_per_iter **do**
    $t \leftarrow t + 1$
    $\ell_\theta(\cdot \mid s_t)$, $\pi_\theta(\cdot \mid s_t) \leftarrow$ ActorNet(s_t) # Forward pass of actor network
    $V_\phi(s_t) \leftarrow$ CriticNet(s_t) # Forward pass of critic network
    Initialize $\pi^m_\theta(\cdot \mid s_t), \tilde{\ell}_\theta(\cdot \mid s_t)$
    Initialize $m(\cdot \mid s_t)$ where $m(a \mid s_t) = 1, \ \forall a \in \mathcal{A}$
    remaining_budget ← yearly_budget
    Shuffle total_components # Shuffle the list of all the components
    for comp ∈ total_components **do**
      if remaining_budget ≥ direct_cost$(a_4,$ comp$)$ **then**
        pass # Action a_4 is affordable: No masking
      else
        if remaining_budget ≥ direct_cost$(a_3,$ comp$)$ **then**
          $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [1, 1, 1, 1, 0]$ # mask a_4
        else
          $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [1, 1, 1, 0, 0]$ # mask a_3, a_4
        end if
      end if
      $\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}} = \begin{cases} \ell_\theta(a_i \mid s_t)_{\text{comp}}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 1 \\ -3.4 \times 10^{38}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 0 \end{cases}$
      $\pi^m_\theta(a_i \mid s_t)_{\text{comp}} = \frac{\exp(\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}})}{\sum_{j=0}^{N} \exp(\tilde{\ell}_\theta(a_j \mid s_t)_{\text{comp}})}$
      $a_{t,\text{comp}} \leftarrow$ sample from $\pi^m_\theta(\cdot \mid s_t)_{\text{comp}}$
      remaining_budget ← remaining_budget − direct_cost$(a_t,$ comp$)$
    end for
    $\pi^m_\theta(a_t \mid s_t) \leftarrow$ evaluate probability of a_t under $\pi^m_\theta(\cdot \mid s_t)$
    $s_{t+1}, R(s_t, a_t) \leftarrow$ environment step

Where: $s_t = [\tilde{s}_t, \ \frac{t}{T}] \ \Rightarrow \ s_{t+1} = [\tilde{s}_{t+1}, \ \frac{t+1}{T}]$

    Store tuple $\{s_t, a_t, \pi^m_\theta(a_t \mid s_t), V_\phi(s_t), R(s_t, a_t), m(\cdot \mid s_t)\}$ in Rollout Memory D_k
    $s_t \leftarrow s_{t+1}$
    if $t = T$ **or** $j =$ steps_per_iter **then**
      if $t = T$ **then**
        $V_\phi(s_{t+1}) \leftarrow 0$
        $s_t \leftarrow$ reset environment
      else
        $V_\phi(s_{t+1}) \leftarrow$ CriticNet(s_{t+1})
      end if
      $\delta_t \leftarrow R(s_t, a_t) + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
      Compute returns: $G_t \leftarrow \sum_{l=0}^{T-t-1} \gamma^l R_{t+l}$
      Compute advantages: $A_t \leftarrow \sum_{l=0}^{T-t-1} (\gamma\tau)^l \delta_{t+l}$
      Store δ_t, A_t in D_k
    end if
  end for
  Train Agent (D_k) using Algorithm 13 # training of networks
end for

Algorithm 16 PPO Rollout with Transferable Yearly Budget Masking

Blue: Deviations from Standard PPO
Red: Deviations from Algorithms 15 & 17

Require: Hyperparameters
yearly_budget ← Load yearly budget
Initialize policy (actor) network weights θ
Initialize value function (critic) network weights ϕ
for $i = 1, \ldots,$ train_iters **do**
  $t \leftarrow 0$, $s_t \leftarrow$ reset environment
  for $j = 1, \ldots,$ steps_per_iter **do**
    $t \leftarrow t + 1$
    $\ell_\theta(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \leftarrow \text{ActorNet}(s_t)$ # Forward pass of actor network
    $V_\phi(s_t) \leftarrow \text{CriticNet}(s_t)$ # Forward pass of critic network
    Initialize $\pi_\theta^m(\cdot \mid s_t), \tilde{\ell}_\theta(\cdot \mid s_t)$
    Initialize $m(\cdot \mid s_t)$ where $m(a \mid s_t) = 1, \ \forall a \in \mathcal{A}$
    remaining_budget $\leftarrow (s_t[-2] - s_t[-1]) \cdot T \cdot$ yearly_budget # Current year's budget including any leftover from previous year
    Shuffle total_components
    for comp ∈ total_components **do**
      if remaining_budget ≥ direct_cost(a_4, comp) **then**
        pass # Action a_4 is affordable: no masking
      else
        if remaining_budget ≥ direct_cost(a_3, comp) **then**
          $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [1, 1, 1, 1, 0]$ # Mask a_4
        else
          $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [1, 1, 1, 0, 0]$ # Mask a_3, a_4
        end if
      end if
      $\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}} = \begin{cases} \ell_\theta(a_i \mid s_t)_{\text{comp}}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 1 \\ -3.4 \times 10^{38}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 0 \end{cases}$
      $\pi_\theta^m(a_i \mid s_t)_{\text{comp}} = \frac{\exp(\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}})}{\sum_{j=0}^{N} \exp(\tilde{\ell}_\theta(a_j \mid s_t)_{\text{comp}})}$
      $a_{t,\text{comp}} \leftarrow$ sample from $\pi_\theta^m(\cdot \mid s_t)_{\text{comp}}$
      remaining_budget $\leftarrow$ remaining_budget − direct_cost($a_{t,\text{comp}}$, comp)
    end for
    $\pi_\theta^m(a_t \mid s_t) \leftarrow$ evaluate probability of a_t under $\pi_\theta^m(\cdot \mid s_t)$
    $s_{t+1}, R(s_t, a_t) \leftarrow$ environment step

Where: $s_t = \left[\tilde{s}_t, \ \frac{t}{T}, \ \beta_t, \ \gamma_t\right] \ \Rightarrow \ s_{t+1} = \left[\tilde{s}_{t+1}, \ \frac{t+1}{T}, \ \frac{t+2}{T}, \ \gamma_t + \frac{\text{direct\_cost}(a_t)}{\text{total\_budget}}\right]$

    Store tuple $\{s_t, a_t, \pi_\theta^m(a_t \mid s_t), V_\phi(s_t), R(s_t, a_t), m(\cdot \mid s_t)\}$ in Rollout Memory D_k
    $s_t \leftarrow s_{t+1}$
    if $t = T$ **or** $j =$ steps_per_iter **then**
      if $t = T$ **then**
        $V_\phi(s_{t+1}) \leftarrow 0$
        $s_t \leftarrow$ reset environment
      else
        $V_\phi(s_{t+1}) \leftarrow \text{CriticNet}(s_{t+1})$
      end if
      $\delta_t \leftarrow R(s_t, a_t) + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
      Compute returns: $G_t \leftarrow \sum_{l=0}^{T-t-1} \gamma^l R_{t+l}$
      Compute advantages: $A_t \leftarrow \sum_{l=0}^{T-t-1} (\gamma\tau)^l \delta_{t+l}$
      Store δ_t, A_t in D_k
    end if
  end for
  Train Agent (D_k) using Algorithm 13 # training of networks
end for

Algorithm 17 PPO Rollout with Global Budget Masking

Blue: Deviations from Standard PPO
Red: Deviations from Algorithms 15 & 16

Require: Hyperparameters
total_budget ← Load total budget
Initialize policy (actor) network weights θ
Initialize value function (critic) network weights ϕ
for $i = 1, \ldots,$ train_iters **do**
  $t \leftarrow 0$, $s_t \leftarrow$ reset environment
  for $j = 1, \ldots,$ steps_per_iter **do**
    $t \leftarrow t + 1$
    $\ell_\theta(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \leftarrow \text{ActorNet}(s_t)$ # Forward pass of actor network
    $V_\phi(s_t) \leftarrow \text{CriticNet}(s_t)$ # Forward pass of critic network
    Initialize $\pi_\theta^m(\cdot \mid s_t), \tilde{\ell}_\theta(\cdot \mid s_t)$
    Initialize $m(\cdot \mid s_t)$ where $m(a \mid s_t) = 1, \ \forall a \in \mathcal{A}$
    remaining_budget $\leftarrow s_t[-1] \cdot$ total_budget # Remaining budget over the full horizon
    Shuffle total_components
    for comp $\in$ total_components **do**
      if remaining_budget $\geq$ direct_cost$(a_4,$ comp$)$ **then**
        pass # Action a_4 is affordable: no masking
      else
        if remaining_budget $\geq$ direct_cost$(a_3,$ comp$)$ **then**
          $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [1, 1, 1, 1, 0]$ # Mask a_4
        else
          $m(\cdot \mid s_t)_{\text{comp}} \leftarrow [1, 1, 1, 0, 0]$ # Mask a_3, a_4
        end if
      end if
      $\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}} = \begin{cases} \ell_\theta(a_i \mid s_t)_{\text{comp}}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 1 \\ -3.4 \times 10^{38}, & \text{if } m(a_i \mid s_t)_{\text{comp}} = 0 \end{cases}$
      $\pi_\theta^m(a_i \mid s_t)_{\text{comp}} = \frac{\exp(\tilde{\ell}_\theta(a_i \mid s_t)_{\text{comp}})}{\sum_{j=0}^{N} \exp(\tilde{\ell}_\theta(a_j \mid s_t)_{\text{comp}})}$
      $a_{t,\text{comp}} \leftarrow$ sample from $\pi_\theta^m(\cdot \mid s_t)_{\text{comp}}$
      remaining_budget $\leftarrow$ remaining_budget $-$ direct_cost$(a_{t,\text{comp}},$ comp$)$
    end for
    $\pi_\theta^m(a_t \mid s_t) \leftarrow$ evaluate probability of a_t under $\pi_\theta^m(\cdot \mid s_t)$
    $s_{t+1}, R(s_t, a_t) \leftarrow$ environment step

Where: $s_t = \left[\tilde{s}_t, \ \frac{t}{T}, \ \frac{\text{remaining\_budget}_t}{\text{total\_budget}}\right] \ \Rightarrow \ s_{t+1} = \left[\tilde{s}_{t+1}, \ \frac{t+1}{T}, \ \frac{\text{remaining\_budget}_t - \text{direct\_cost}(a_t)}{\text{total\_budget}}\right]$

    Store tuple $\{s_t, a_t, \pi_\theta^m(a_t \mid s_t), V_\phi(s_t), R(s_t, a_t), m(\cdot \mid s_t)\}$ in Rollout Memory D_k
    $s_t \leftarrow s_{t+1}$
    if $t = T$ **or** $j =$ steps_per_iter **then**
      if $t = T$ **then**
        $V_\phi(s_{t+1}) \leftarrow 0$
        $s_t \leftarrow$ reset environment
      else
        $V_\phi(s_{t+1}) \leftarrow \text{CriticNet}(s_{t+1})$
      end if
      $\delta_t \leftarrow R(s_t, a_t) + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
      Compute returns: $G_t \leftarrow \sum_{l=0}^{T-t-1} \gamma^l R_{t+l}$
      Compute advantages: $A_t \leftarrow \sum_{l=0}^{T-t-1} (\gamma\tau)^l \delta_{t+l}$
      Store δ_t, A_t in D_k
    end if
  end for
  Train Agent (D_k) using Algorithm 13 # training of networks
end for

6.5 Lagrangian Relaxation

While Action Masking is highly effective for constraints that can be evaluated at the component level (such as failure probability) or purely additive resources (such as budgets), it is ill-suited for the Maximum City Costs constraint defined in Section 6.1.

City costs are non-linear and global; the cost of closing a street depends on the simultaneous status of adjacent streets. Consequently, determining whether a specific action is "valid" via masking would require pre-calculating the total city cost for every possible combination of actions across the entire network at every timestep, which is computationally intractable. Furthermore, unlike strict budgetary limits, city costs are often treated as "soft" constraints in practice—targets that should be met on average or in aggregate over the planning horizon, rather than physical impossibilities at a single instant.

To rigorously enforce such global constraints without the computational burden of exhaustive pre-calculation, we adopt Lagrangian Relaxation. Instead of filtering actions a priori, this method shifts the burden of constraint satisfaction to the optimization objective itself.

Formally, Lagrangian relaxation is a mathematical optimization technique used to solve constrained problems by transforming them into an unconstrained dual form. Specifically, the method "relaxes" the hard constraints by incorporating them directly into the objective function via Lagrange multipliers. This reformulation allows the problem to be solved as a saddle-point optimization, where the policy attempts to maximize rewards while simultaneously minimizing the penalty for constraint violations.

Consider the constrained reinforcement learning problem where the agent aims to maximize an expected return subject to one or more constraints (Equation (77)):

$$\text{maximize}_\theta \quad J_r(\pi_\theta) \quad \text{subject to} \quad J_c(\pi_\theta) \leq d \tag{77}$$

Where $J_r(\pi_\theta)$ denotes the expected return (Equation (78)) and $J_c(\pi_\theta)$ denotes the expected cost (Equation (79)), with upper bound d.

$$J_r(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \tag{78}$$

$$J_c(\pi_\theta) = \mathbb{E}_{\pi_\theta}\big[C(s_t)\big] \tag{79}$$

The constraint function $C(s_t)$ is defined as $C(s_t) = F\big(c(s_t, a_t), \ldots, c(s_N, a_N)\big)$, where the specific form of $F(\cdot)$ depends on the constraint being imposed. For example, the constraint may take the form of a discounted cumulative sum ($C(s_t) = \sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)$), an average cost ($C(s_t) = \frac{1}{N}\sum_{t=0}^{N} c(s_t, a_t)$), or other variants (see [52] for additional examples).

Lagrangian relaxation reformulates this constrained problem by defining the **Lagrangian function**, as shown in Equation (80).

$$\mathcal{L}(\theta, \lambda) = J_r(\pi_\theta) - \lambda\big[J_c(\pi_\theta) - d\big] \tag{80}$$

Where $\lambda \geq 0$ is a non-negative scalar (or vector, if there are multiple constraints) representing the Lagrange multipliers. Intuitively:

- If $\lambda = 0$, the constraint is ignored;
- If λ increases, the penalty for violating the constraint grows, enforcing it more strictly.

The optimization then becomes a **saddle-point problem** (see Equation (81)).

$$\min_{\lambda \geq 0} \max_{\theta} \mathcal{L}(\theta, \lambda) \tag{81}$$

Where the policy parameters θ are optimized to maximize the Lagrangian while the multipliers λ are adjusted to minimize it, ensuring constraint satisfaction.

In reinforcement learning, this formulation introduces a trade-off between maximizing reward and satisfying constraints. There are multiple approaches to implement this:

- **Naive fixed-penalty methods:** One could manually tune a fixed λ and optimize $\mathcal{L}(\theta, \lambda)$ using standard PPO. While conceptually inspired by the Lagrangian idea, this is not strictly Lagrangian relaxation, as λ is not adapted based on constraint violations. Fixed-penalty methods are simple but brittle:
  - If λ is too small, the agent may ignore the constraint, leading to violations;
  - If λ is too large, the agent over-penalizes, sacrificing reward unnecessarily.
- **Adaptive Lagrangian methods:** Modern Lagrangian relaxation methods treat λ as a learnable dual variable that is updated alongside the policy. A typical update uses projected stochastic gradient ascent on λ (see Equation (82)).

  $$\lambda_{k+1} = \Big[\lambda_k + \alpha_\lambda \big(J_c(\pi_{\theta_k}) - d\big)\Big]_+ \tag{82}$$

  Where $\alpha_\lambda > 0$ is the learning rate for the multiplier, and $[\cdot]_+$ denotes projection onto the non-negative orthant to enforce $\lambda \geq 0$.

Although adaptive Lagrangian methods can be sensitive to hyperparameters (such as α_λ) and may oscillate during training, they provide a systematic mechanism to balance reward maximization with constraint satisfaction. Empirical studies in safe RL show that careful stabilization of multiplier updates (e.g., using PID-inspired control [53]) can significantly reduce constraint violations during learning.

Implementing Lagrangian relaxation on top of PPO is not trivial, and there are multiple ways of integrating the safety signals. In the literature, two primary architectures have

been proposed for applying Lagrangian relaxation on top of PPO: Reward Constrained Policy Optimization (RCPO) [54] and PPO-Lagrangian (PPO-Lag) [55]. RCPO operates via scalarization at the reward level, fundamentally altering the agent's perception of "value", while PPO-Lagrangian operates at the gradient update level. In this work, both approaches were evaluated. However, stability limitations (discussed in the following subsection) led to RCPO being excluded from the final experimental study, and PPO-Lagrangian was adopted as the primary mechanism for enforcing soft constraints.

6.6 Reward Constrained Policy Optimization

Reward Constrained Policy Optimization (RCPO) is a practical method for incorporating constraints directly into policy optimization in reinforcement learning at the signal level. Building on the Lagrangian relaxation framework, RCPO modifies the policy update by scalarizing the constrained objective into a single surrogate reward function (Equation (83)) [54].

$$\hat{R}(s, a; \lambda) = R(s, a) - \lambda \cdot c(s, a) \tag{83}$$

Which is then optimized using standard policy gradient methods such as PPO. Unlike naive fixed-penalty methods, RCPO adapts the Lagrange multiplier dynamically during training, providing a principled mechanism to enforce constraints without requiring extensive manual tuning. The most common method described in the literature [54] is to update the multiplier using stochastic gradient ascent (Equation (82)).

Even though replacing the original reward with a penalized reward may appear to be a minor modification, this change fundamentally alters the learning dynamics of the algorithm. In RCPO, the policy is optimized with respect to a *penalized reward*, which directly affects the value function, temporal-difference targets, return estimates, advantage estimates, and, consequently, both the actor and critic updates.

Specifically, the value function under the penalized reward can be written as shown in Equation (84).

$$\begin{aligned} \hat{V}^{\pi}(s; \lambda) &= \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \hat{R}(s_t, a_t; \lambda) \,\middle|\, s_0 = s \right] \\ &= \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \big(R(s_t, a_t) - \lambda c(s_t, a_t) \big) \,\middle|\, s_0 = s \right] \end{aligned} \tag{84}$$

This decomposition shows that RCPO implicitly learns a value function that trades off reward maximization against constraint satisfaction through the Lagrange multiplier λ.

The use of the penalized reward also modifies the temporal-difference (TD) target. Which given a value function approximation $\hat{V}_{\phi}$, is computed using Equation (85).

$$\hat{\delta}_t^{\lambda} = \hat{R}(s_t, a_t; \lambda) + \gamma \hat{V}_{\phi}(s_{t+1}) - \hat{V}_{\phi}(s_t) \tag{85}$$

Similarly, the discounted return becomes defined by Equation (86), and the advantage function by Equation (87).

$$\hat{G}_t^{\lambda} = \sum_{l=0}^{\infty} \gamma^l \hat{R}(s_{t+l}, a_{t+l}; \lambda) \tag{86}$$

$$\hat{A}_t^{\lambda} = \sum_{l=0}^{\infty} (\gamma\tau)^l \hat{\delta}_{t+l}^{\lambda} \tag{87}$$

As a consequence, both the actor and critic losses in PPO are modified. The actor loss is computed using the penalized advantage (Equation (88)) and the critic using the penalized return (Equation (89)).

$$\mathcal{L}_{\text{actor}}^{\text{RCPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\, \hat{A}_t^{\lambda},\ \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\, \hat{A}_t^{\lambda} \right) + c_2\, H\big[\pi_\theta(\cdot \mid s_t)\big] \right] \tag{88}$$

$$\mathcal{L}_{\text{critic}}^{\text{RCPO}}(\phi) = \mathbb{E}_t \Big[\big(\hat{V}_\phi(s_t) - \hat{G}_t^{\lambda} \big)^2 \Big] \tag{89}$$

As we can see, RCPO propagates the effect of constraint violations throughout the entire learning pipeline by embedding the Lagrangian penalty directly into the reward signal (see Algorithm 18). While this enables straightforward integration with PPO, it tightly couples reward learning and constraint enforcement. Consequently, variations in the Lagrange multiplier λ directly modify the effective reward distribution, altering value estimation, advantage computation, and critic targets.

In practice, this introduces strong non-stationarity in critic learning, which negatively affected training stability during preliminary evaluations. Based on these observations, RCPO was not pursued further in this work. Instead, PPO-Lagrangian was adopted as a more robust alternative that decouples reward and cost learning while preserving Lagrangian constraint enforcement.

Algorithm 18 Reward Constrained Policy Optimization (RCPO)

Require: Initial policy parameters θ_0, value function parameters ϕ_0, initial multiplier $\lambda_0 \geq 0$
Require: Number of iterations K, clipping parameter ϵ
Require: Learning rates α_θ, α_ϕ, α_λ
1: **for** $k = 0, 1, \ldots, K-1$ **do**
2: Collect set of trajectories $\mathcal{D}_k = \{\nu_i\}$ by running policy π_{θ_k}
3: Compute penalized rewards:

$$\hat{R}_t = R(s_t, a_t) - \lambda_k c(s_t, a_t)$$

4: Estimate penalized returns $\hat{G}_t^\lambda = \sum_{l=0}^{\infty} \gamma^l \hat{R}(s_{t+l}, a_{t+l}; \lambda)$
5: Estimate penalized advantages $\hat{A}_t^\lambda = \sum_{l=0}^{\infty} (\gamma\tau)^l \hat{\delta}_{t+l}^\lambda$
6: Update policy parameters by maximizing:

$$\theta_{k+1} = \arg\max_\theta \mathbb{E}_t \left[\min\left(r_t(\theta)\, \hat{A}_t^\lambda,\ \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\, \hat{A}_t^\lambda \right) + c_2\, H\big[\pi_\theta(\cdot \mid s_t)\big] \right]$$

7: Update value function parameters by regression:

$$\phi_{k+1} = \arg\min_\phi \mathbb{E}_t \left[\left(\hat{V}_\phi(s_t) - \hat{G}_t^\lambda \right)^2 \right]$$

8: Update Lagrange multiplier (projected ascent):

$$\lambda_{k+1} = \Big[\lambda_k + \alpha_\lambda \big(J_c(\pi_{\theta_k}) - d \big) \Big]_+$$

9: **end for**

6.7 PPO-Lagrangian

PPO-Lagrangian [55] represents a structural evolution of RCPO. Rather than scalarizing reward and cost at the input level, PPO-Lagrangian preserves their separation throughout the value estimation process and integrates them only at the policy gradient computation stage. This design allows reward and cost signals to be learned independently while still enforcing safety constraints during policy optimization.

To enable this separation, PPO-Lagrangian introduces the Lagrange multiplier λ at the advantage level instead of directly modifying the reward function. In contrast to RCPO, where the reward is explicitly penalized by subtracting λc_t at each timestep (Equation (83)), PPO-Lagrangian maintains the original reward and cost definitions during value learning. Separate advantage functions are computed for reward and cost, and the penalty is applied only when forming the combined advantage used in the policy gradient. The resulting Lagrangian advantage is defined in Equation (90).

$$A_t^{\text{Lag}} = A_t^R - \lambda A_t^C \tag{90}$$

A_t^R and A_t^C denote the reward and cost advantages, respectively. This formulation ensures that policy updates reflect the current constraint pressure imposed by λ, while keeping reward and cost learning decoupled and statistically well-behaved.

6.7.1 Architecture: Dual-Critic System

To support separate advantage estimation, PPO-Lagrangian employs a dual-critic architecture (Figure 12) consisting of a reward critic and a cost critic:

- The reward critic $V_R(s)$ estimates the expected discounted cumulative task reward (Equation (91)).

$$V_R(s_t) \approx \mathbb{E}_{\pi_\theta}\left[\sum_{l=0}^{\infty} \gamma^l R_{t+l} \mid s_t\right] \tag{91}$$

- The cost critic $V_C(s)$ estimates the expected discounted cumulative safety cost (Equation (92)).

$$V_C(s_t) \approx \mathbb{E}_{\pi_\theta}\left[\sum_{l=0}^{\infty} \gamma^l c_{t+l} \mid s_t\right] \tag{92}$$

Each critic is trained using a separate Mean Squared Error (MSE) loss. The reward critic minimizes the loss defined in Equation (93), while the cost critic minimizes the loss in Equation (94).

$$\mathcal{L}_{V_R} = \mathbb{E}_{s_t \sim \pi_\theta}\left[\left(V_R(s_t) - G_t^R\right)^2\right] \tag{93}$$

$$\mathcal{L}_{V_C} = \mathbb{E}_{s_t \sim \pi_\theta}\left[\left(V_C(s_t) - G_t^C\right)^2\right] \tag{94}$$

Where the reward return G_t^R and cost return G_t^C used as regression targets are defined in Eqs. (95) and (96), respectively:

$$G_t^R = \sum_{l=0}^{\infty} \gamma^l R_{t+l} \tag{95}$$

$$G_t^C = \sum_{l=0}^{\infty} \gamma^l c_{t+l} \tag{96}$$

By decoupling value estimation in this manner, PPO-Lagrangian effectively mitigates the non-stationarity issues inherent in RCPO. Since both the reward function $R(s, a)$ and the cost function $C(s, a)$ are static properties of the environment, the learning targets for V_R and V_C remain stationary (assuming a slowly evolving policy). As a result, even if the Lagrange multiplier λ oscillates significantly during training, the reward critic continues to learn the true structure of the task reward and does not "forget" valuable states due to transient constraint pressure.

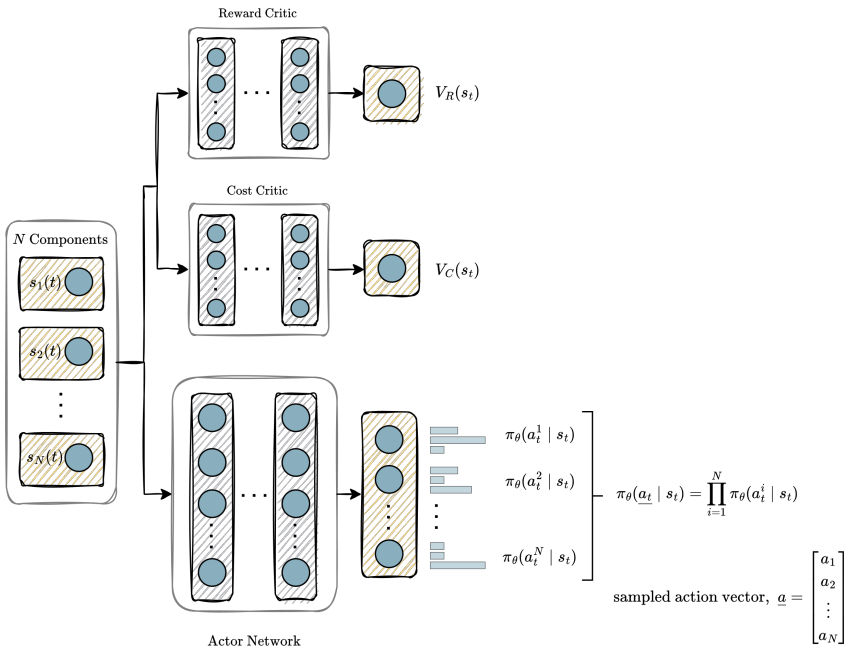

Figure 12: Dual-critic DCMAC architecture

6.7.2 Advantage Estimation

Finally, PPO-Lagrangian computes advantages using separate GAE procedures for reward and cost. The reward advantage A_t^R and the cost advantage A_t^C are defined, respectively, in Equation (97) and Equation (98).

$$A_t^R = \sum_{l=0}^{\infty} (\gamma \lambda_R)^l \delta_{t+l}^R \tag{97}$$

$$A_t^C = \sum_{l=0}^{\infty} (\gamma \lambda_C)^l \delta_{t+l}^C \tag{98}$$

Where the reward temporal-difference (TD) error is given by Equation (99) and the cost TD error is defined in Equation (100).

$$\delta_t^R = R_t + \gamma V_R(s_{t+1}) - V_R(s_t) \tag{99}$$

$$\delta_t^C = c_t + \gamma V_C(s_{t+1}) - V_C(s_t) \tag{100}$$

Here, λ_R and λ_C denote the GAE bias–variance trade-off parameters for reward and cost, respectively, and are often set to the same value in practice. These separate advantage estimators preserve the statistical structure of reward and cost signals, while allowing the

Lagrangian penalty to be incorporated only at the policy optimization stage via the combined advantage defined in Equation (90).

To provide a clear overview of the full optimization workflow, the PPO-Lagrangian training procedure is summarized in Algorithm 19. The algorithm outlines how trajectory collection, dual-critic value estimation, Lagrange multiplier adaptation, and clipped-surrogate policy updates are performed iteratively during training.

Algorithm 19 PPO-Lagrangian

Require: Initial policy parameters θ_0
Require: Reward critic parameters ϕ_R^0, cost critic parameters ϕ_C^0
Require: Initial multiplier $\lambda_0 \geq 0$
Require: Number of iterations K, clipping parameter ϵ
Require: Learning rates α_θ, α_{ϕ_R}, α_{ϕ_C}, α_λ

1: **for** $k = 0, 1, \ldots, K-1$ **do**
2: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy π_{θ_k}
3: Collect rewards and costs: $R(s_t, a_t)$, $c(s_t, a_t)$
4: Estimate reward returns G_t^R and cost returns G_t^C:

$$G_t^R = \sum_{l=0}^{\infty} \gamma^l R_{t+l}, \quad G_t^C = \sum_{l=0}^{\infty} \gamma^l c_{t+l}$$

5: Compute reward and cost TD-errors:

$$\delta_t^R = R_t + \gamma V_R(s_{t+1}) - V_R(s_t), \quad \delta_t^C = c_t + \gamma V_C(s_{t+1}) - V_C(s_t)$$

6: Estimate reward and cost advantages using GAE:

$$A_t^R = \sum_{l=0}^{\infty} (\gamma\lambda_R)^l \delta_{t+l}^R, \quad A_t^C = \sum_{l=0}^{\infty} (\gamma\lambda_C)^l \delta_{t+l}^C$$

7: Update Lagrange multiplier (projected ascent):

$$\lambda_{k+1} = \Big[\lambda_k + \alpha_\lambda \big(J_c(\pi_{\theta_k}) - d\big)\Big]_+$$

8: Form Lagrangian advantage:

$$A_t^{\text{Lag}} = A_t^R - \lambda_k A_t^C$$

9: Update policy parameters by maximizing:

$$\theta_{k+1} = \arg\max_\theta \mathbb{E}_t \left[\min\Big(r_t(\theta)\, A_t^{\text{Lag}},\ \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\, A_t^{\text{Lag}} \Big) + c_2\, H\big[\pi_\theta(\cdot \mid s_t)\big] \right]$$

10: Update reward critic by regression:

$$\phi_R^{k+1} = \arg\min_{\phi_R} \mathbb{E}_t \left[\big(V_R(s_t) - G_t^R\big)^2 \right]$$

11: Update cost critic by regression:

$$\phi_C^{k+1} = \arg\min_{\phi_C} \mathbb{E}_t \left[\big(V_C(s_t) - G_t^C\big)^2 \right]$$

12: **end for**

6.7.3 Practical implementation details

I. Lagrangian Scaling

While the canonical formulation of PPO-Lagrangian defines the combined advantage as a linear combination of reward and cost advantages (Equation (90)), many practical implementations introduce an additional scaling step. In particular, widely used safe reinforcement learning libraries such as OmniSafe [56] and SafePO [57] scale the total advantage by a factor of $(1+\lambda)^{-1}$ to improve numerical stability when the Lagrange multiplier becomes large. Under this modification, the combined advantage used for policy optimization is given by Equation (101):

$$A_t^{\text{Lag}} = \frac{A_t^R - \lambda A_t^C}{1+\lambda} \tag{101}$$

This scaling does not alter the qualitative structure of the Lagrangian objective; rather, it introduces an explicit normalization with respect to the current value of the multiplier λ. The primary motivation for this adjustment is numerical and optimization stability. In the standard PPO-Lagrangian formulation, large values of λ can cause the magnitude of the combined advantage $A_t^{\text{Lag}} = A_t^R - \lambda A_t^C$ to grow excessively.

Because PPO's clipped surrogate objective is sensitive to the scale of the advantage estimates, such variations can result in overly aggressive or erratic policy updates, even in the presence of clipping. Dividing the combined advantage by $1+\lambda$ ensures that its overall magnitude remains bounded regardless of how large the multiplier becomes. Effectively, this normalization acts as an adaptive gain control mechanism: as constraint pressure increases and λ grows, the policy gradient continues to emphasize cost minimization, but the effective update step is implicitly damped.

As a result, Lagrangian scaling improves robustness to noisy or rapidly changing multiplier values and reduces oscillatory behavior during training. For these reasons, although not part of the original theoretical formulation, this scaled advantage has become a common practical refinement in modern PPO-Lagrangian implementations.

II. Advantage normalization

In addition to Lagrangian scaling, practical PPO-Lagrangian implementations often apply normalization to advantage estimates prior to policy optimization. Advantage normalization is a widely adopted technique in PPO to stabilize gradient updates by ensuring that advantages have approximately zero mean and unit variance within each training batch. Without normalization, large variations in advantage magnitude can lead to unstable or excessively large policy updates, particularly when reward and cost signals differ significantly in scale.

In the present work, reward and cost advantages are first computed independently using GAE. Each advantage stream is then normalized across the collected batch, yielding normalized reward and cost advantages, denoted $\hat{A}_t^R$ and $\hat{A}_t^C$, respectively. The normalized

advantages are computed as shown in Equation 102:

$$\hat{A}_t^R = \frac{A_t^R - \mu_R}{\sigma_R + \varepsilon}, \qquad \hat{A}_t^C = \frac{A_t^C - \mu_C}{\sigma_C + \varepsilon}, \tag{102}$$

where μ_R and μ_C denote the batch means, σ_R and σ_C denote the batch standard deviations, and ε is a small constant introduced for numerical stability.

III. Combined normalization and Lagrangian scaling

When both advantage normalization and Lagrangian scaling are applied, the final advantage used for policy optimization is defined as shown in Equation 103.

$$\hat{A}^{\text{Lag}} = \frac{\hat{A}_t^R - \lambda \hat{A}_t^C}{1 + \lambda} \tag{103}$$

This formulation combines three stabilization mechanisms: independent normalization of reward and cost advantages, adaptive weighting through the Lagrange multiplier, and magnitude control via scaling with $(1 + \lambda)^{-1}$. Together, these modifications help ensure that policy updates remain numerically stable while still responding appropriately to constraint violations. This final advantage formulation is used throughout all experiments in this work.

6.7.4 Sensitivity of Lagrange Multiplier Updates

While PPO-Lagrangian significantly improves stability over reward-level scalarization methods such as RCPO by decoupling reward and cost value estimation, it does not completely eliminate non-stationarity from the learning process. Although the separation of critics ensures that both the reward critic and the cost critic learn stationary targets corresponding to fixed environment signals, the policy optimization problem itself remains inherently non-stationary due to the adaptive nature of the Lagrange multiplier.

In particular, the actor in PPO-Lagrangian optimizes a Lagrangian objective whose form depends explicitly on the current value of the multiplier λ. Each update to λ alters the relative weighting between reward maximization and constraint satisfaction in the policy gradient. Consequently, updates to the multiplier effectively shift the location of the policy's local optimum over time. If λ is adjusted too aggressively or on an inappropriate timescale, the policy may repeatedly pursue a moving objective, leading to oscillatory behavior or stalled convergence even when the critics themselves remain stable.

In its standard formulation, PPO-Lagrangian updates the Lagrange multiplier using projected stochastic gradient ascent (Equation (82)), mirroring the update rule commonly employed in RCPO. While this approach is simple, theoretically grounded, and straightforward to implement, empirical studies have shown that it can be overly sensitive to noisy cost estimates [53]. As a result, the multiplier may oscillate around its optimal value, slowing policy convergence and degrading constraint satisfaction performance.

To mitigate these issues, more adaptive multiplier update mechanisms have been proposed. Notably, Stooke et al. [53] introduced a control-theoretic approach that frames the Lagrange multiplier update as a proportional–integral–derivative (PID) control problem. This method explicitly incorporates information about the magnitude, accumulation, and rate of change of constraint violations, enabling more stable and responsive multiplier dynamics. A detailed description of this PID-based enhancement to PPO-Lagrangian is provided in the following section.

6.8 PID Lagrangian

While standard PPO-Lagrangian mitigates some instabilities by separating reward and cost critics, the Lagrange multiplier update itself remains reactive and purely integral, responding only to accumulated constraint violations. As a consequence, the multiplier often adjusts too slowly: it continues to increase or decrease even after the optimal correction has been reached. This delayed response leads to oscillatory behavior, commonly referred to as "overshooting" or "bang-bang control", where policies alternate between unsafe and overly conservative actions and rarely settle on the true constraint boundary.

To address this limitation, PID-Lagrangian extends the traditional update with proportional and derivative terms. By reacting to the current error, considering the accumulated history, and anticipating the rate of change, PID-Lagrangian produces damped and predictive multiplier dynamics, substantially reducing oscillations in constraint violations and improving convergence to the desired safety boundary.

6.8.1 PID Controller

A proportional–integral–derivative (PID) controller is a classical feedback mechanism widely used in control systems to regulate a process toward a target setpoint. It computes the error $e(t)$ between a desired target and a measured variable, then adjusts the control input $u(t)$ as a weighted sum of three terms: proportional, integral, and derivative [58] (see Figure 13).

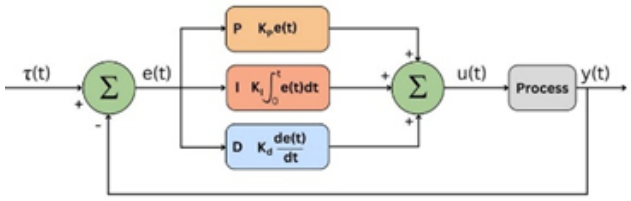

Figure 13: Illustration of a PID controller [5].

- **Proportional term (K_P)** [Figure 14a]: Reacts to the current error, improving immediate responsiveness and reducing constraint violations quickly;

- **Integral term (K_I)** [Figure 14b]: Accumulates past errors to eliminate steady-state bias, correcting persistent deviations that proportional control alone cannot remove;
- **Derivative term (K_D)** [Figure 14c]: Responds to the rate of change of the error, providing anticipatory damping that reduces overshoot and stabilizes updates.

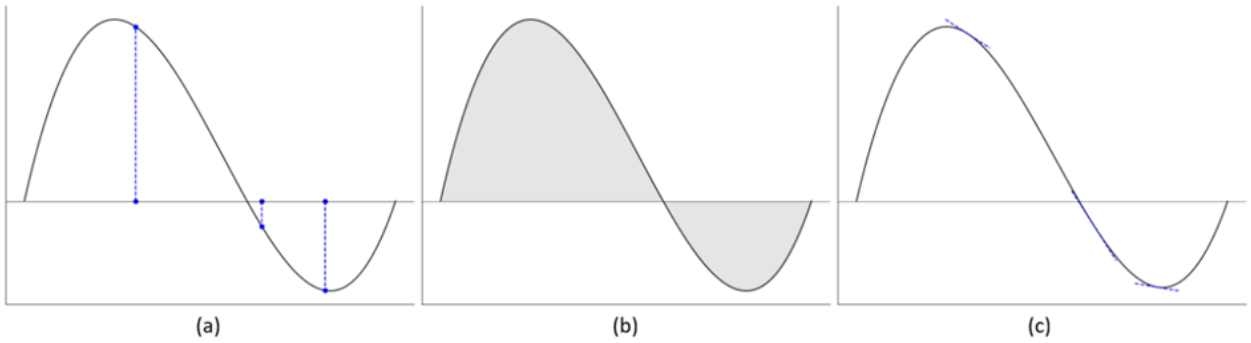

Figure 14: Contribution of each PID term to control action. Adapted from [6].

6.8.2 PID-Lagrangian as a Control Mechanism

In constrained reinforcement learning, the PID controller can be interpreted as regulating policy safety via the Lagrange multiplier:

- The **error (e_k)** entering the PID controller is the difference between the expected cost of policy π, $J_C(\pi)$, and the constraint limit d (Equation (104)).

$$e_k = J_C(\pi_k) - d \tag{104}$$

- The **control input (u_k)** is the Lagrange multiplier λ, which adjusts policy behavior to satisfy constraints. The discrete-time PID update is given by Equation (105):

$$\lambda_k = \left[K_P e_k + K_I \sum_{i=0}^{k} e_i + K_D \left(e_k - e_{k-1} \right) \right]_+ \tag{105}$$

  Where $[\cdot]_+$ ensures non-negativity of λ, and $K_P, K_I, K_D \geq 0$ are the PID gains.

- The **output (y_k)** is the observed cost $J_C(\pi)$.

Our PID update rule (Equation (105)), which replaces the standard Lagrange multiplier update (step 7 in Algorithm 19), is summarized in Algorithm 20.

Algorithm 20 PID-Controlled Lagrange Multiplier

Require: Constraint limit l
Require: PID gains $K_P, K_I, K_D \geq 0$
Ensure: Lagrange multiplier $\lambda \geq 0$
Initialize multiplier $\lambda \leftarrow 0$
Initialize integral term $I \leftarrow 0$
Initialize previous cost $J_{c,\text{prev}} \leftarrow l$
for each iteration k **do**
    Observe expected cost J_c^π
    Compute constraint deviation: $e \leftarrow J_c^\pi - l$
    Update integral term: $I \leftarrow (I + e)_+$
    Compute derivative term: $d \leftarrow (J_c^\pi - J_{c,\text{prev}})_+$
    Update multiplier:

$$\lambda \leftarrow [K_P e + K_I I + K_D d]_+$$

    Store previous cost: $J_{c,\text{prev}} \leftarrow J_c^\pi$
end for
return λ

Unlike pure gradient-ascent updates, the PID-based multiplier reacts to instantaneous violations, corrects persistent bias, and anticipates future trends, resulting in smoother and more reliable safety regulation. Empirical studies show that PID-Lagrangian substantially improves both training stability and constraint satisfaction across standard benchmark tasks [53]. For these reasons, we adopt PID-Lagrangian to implement the final constraint in this project.

6.9 Curriculum Learning

The constraint-handling mechanisms described in Sections 6.3 and 6.8 provide the algorithmic tools necessary to enforce validity. However, having a method to enforce constraints does not guarantee that a feasible policy can be learned efficiently. In high-dimensional environments with strict constraints, the feasible region of the policy space can be sparse and difficult to discover from a random Neural Network initialization. An agent training from scratch may struggle to find any valid sequence of actions, leading to vanishing gradients, slow convergence, or entrapment in poor local optima.

Curriculum learning, as formalized by Bengio et al. [59], proposes that training machine learning models in a structured progression of increasing difficulty can improve convergence speed and generalization. In the context of DRL, this translates to first training an agent in a simpler version of the environment and gradually scaling to the full, complex environment. By doing so, the agent can leverage knowledge acquired in simpler settings to efficiently explore and optimize in more challenging ones.

In our actor-critic framework, curriculum learning is implemented through a warm-start strategy. Specifically, the agent is first trained in a simpler environment, after which the learned neural network parameters of both the actor and the critic are used to initialize

training in the more complex environment:

$$\theta_{k+1}^{\text{actor}} \leftarrow \theta_{k}^{\text{actor}}$$

$$\phi_{k+1}^{\text{critic}} \leftarrow \phi_{k}^{\text{critic}}$$

This allows the agent to start from a more informed state rather than from random initialization. For networks that exist in both environments, weights and biases are directly transferred.

In this thesis, curriculum learning is applied in two primary ways:

Introducing constraints to an unconstrained agent: If an agent has already been trained in a large, unconstrained environment, introducing constraints may make finding an optimal policy more challenging. In this case, we can warm-start the constrained agent using the parameters of the unconstrained one. This approach is effective if the optimal actions under the new constraints do not differ significantly from the unconstrained optimum. To mitigate the risk of local minima, it may be necessary to temporarily increase the entropy coefficient, allowing the agent to "unlearn" certain behaviors, explore its surroundings more freely, and escape suboptimal policies.

Scaling up the environment: Once an agent has successfully learned a constrained policy in a smaller environment, its parameters can be used to warm-start training in a larger environment. It is crucial that components from the smaller environment are preserved and placed consistently in the larger setup so that the corresponding nodes and structures align correctly with the previously learned policy.

Both strategies will be explored in this thesis, combining curriculum learning with action masking to enhance the efficiency and stability of constrained DRL training.

7 Experimental Setup and Results

In this section, we describe the various experiments conducted and evaluate the results obtained throughout the thesis using the methodologies introduced in the previous chapters. The main objective is to assess the performance, correctness, and scalability of the proposed DRL–based maintenance planning framework, both in the unconstrained setting and under the different constraint-handling techniques developed in this work.

The goals of this section are threefold. First, we evaluate the performance of the DRL agent on the unconstrained problem in order to establish a reliable baseline and verify stable learning behavior. Second, we benchmark the learned policies against rule-based baselines and a mathematical optimization approach based on the CEM, allowing us to assess both solution quality and scalability. Third, we evaluate the correct implementation and practical impact of the different constraint-handling techniques integrated into PPO, including the effect of curriculum learning as a warm-start strategy for constrained training.

All experiments are conducted over a fixed planning horizon of 20 timesteps, with the environment simulated deterministically. Rather than sampling discrete outcomes from the stochastic transition matrices at each step, the environment state propagates the exact probability distribution of each component across all condition states. This setup enables a consistent comparison of policies across different configurations without the variance introduced by random sampling, while still preserving the essential stochastic system dynamics. Prior to introducing explicit constraints, a sensitivity analysis is performed by varying the environment's cost weights. This analysis demonstrates how policy behavior can be shaped through modifications to the reward structure and environment dynamics, without imposing explicit constraints or restricting the feasible action space.

Unless stated otherwise, the results in this section are based on a subsystem of 16 quay wall sections (Subsystem 1), corresponding to three complete quay-wall groups. This configuration balances practicality and realism: it is small enough to allow extensive experimentation and detailed analysis, yet sufficiently complex to capture the key dynamics of the real system. In particular, selecting complete quay-wall groups is essential to correctly model city-level costs and interdependencies between components. Using three full groups therefore provides a balanced trade-off: the problem remains computationally manageable, yet still reflects the coupled, network-level behavior characteristic of urban infrastructure systems.

Most validation and benchmarking results are therefore reported for this environment size. Additional experiments were conducted for other environment sizes following the same experimental procedure. To avoid unnecessary repetition and overloading the reader with redundant information, only the most representative results are presented and discussed in detail.

7.1 Unconstrained Problem: Baseline Performance

We begin by analyzing the unconstrained version of the problem. This step is necessary to validate the basic functioning of the DRL framework before introducing any constraints. In particular, we aim to verify that the agent can be trained successfully, that learning is stable and convergent, and that the resulting policy achieves competitive performance relative to established decision-making strategies.

Beyond confirming convergence, it is essential to assess the quality of the learned policy. As introduced in Chapter 5, the proposed DRL approach is benchmarked against a set of rule-based and reactive strategies that reflect current practice in infrastructure maintenance planning for municipalities such as Amsterdam. While outperforming such heuristics does not guarantee optimality, it provides an important first indication that the agent learns meaningful and effective decision rules.

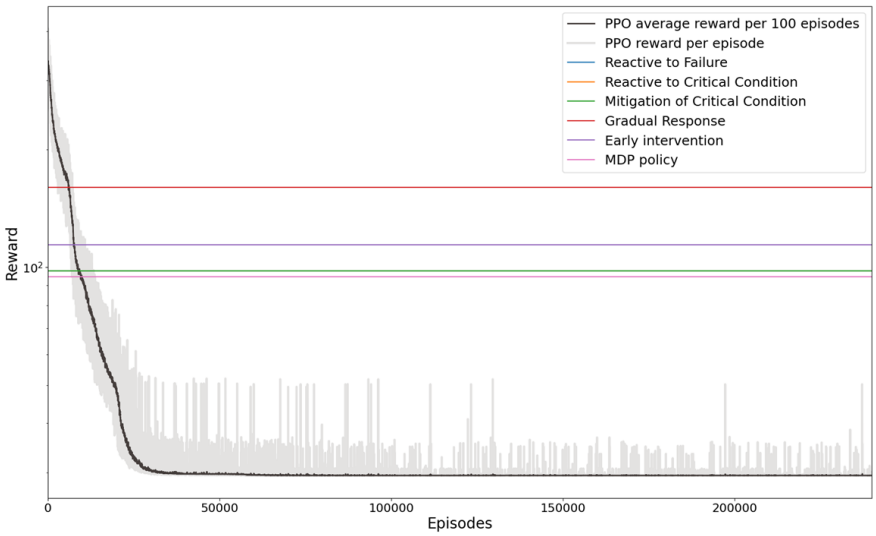

Figure 15: Training of the DRL agent and comparison with the reward obtained by benchmark strategies.

Figure 15 shows the training curve of the DRL agent together with the returns obtained by the benchmark strategies. The agent converges reliably and achieves a significantly higher total return than all benchmark methods. A quantitative comparison is provided in Table 9, which reports the total expected cost achieved by each approach, as well as the ratio relative to the DRL agent's performance.

	Total expected cost	Total expected cost divided by DRL total expected cost
Reactive to Failure	-98.0 M €	3.33
Reactive to Critical Condition	-98.0 M €	3.33
Mitigation of Critical Condition	-98.0 M €	3.33
Gradual Response	-160.1 M €	5.45
Early Intervention	-114.2 M €	3.88
Pseudo MDP policy	-94.7 M €	3.22
Actor Network	**-29.4 M €**	1.00

Table 9: Comparison of total expected costs achieved by the DRL agent and benchmark strategies.

The results show that the DRL agent consistently outperforms all considered baselines. Reactive and rule-based strategies yield total expected costs that are more than three times higher than those obtained by the DRL agent. This confirms that the proposed learning-based approach is capable of exploiting the structure of the problem more effectively than static or myopic decision rules.

A more detailed breakdown of the different cost components incurred by both the DRL agent and the benchmark strategies is provided in Figure 16.

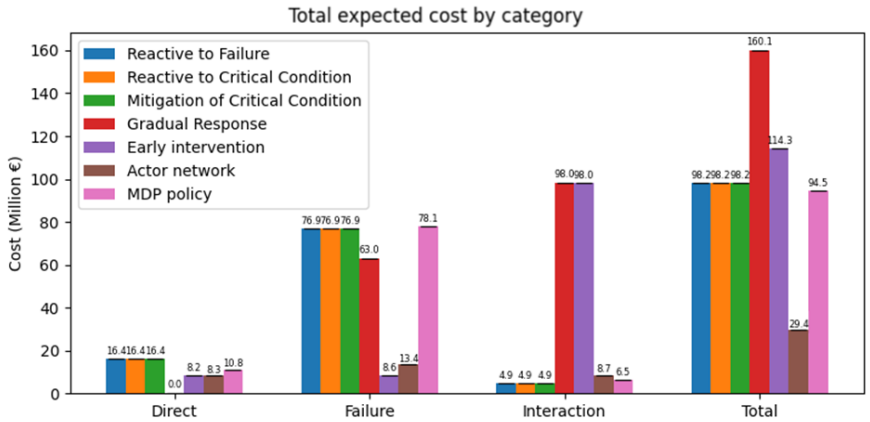

Figure 16: Breakdown of the costs for the DRL agent and benchmark strategies.

Taken together, these results demonstrate that the DRL agent can be trained successfully on the unconstrained problem, converges to a stable solution, and achieves substantially better performance than commonly used benchmark strategies. However, as discussed previously, outperforming heuristic baselines does not provide guarantees regarding optimality. For this reason, a more rigorous validation of the model against a mathematical optimization method is required.

7.2 Validation Against the Cross-Entropy Method

To further validate the proposed DRL approach, we benchmark it against the CEM, a stochastic, simulation-based optimization technique introduced in Chapter 5. CEM directly optimizes action sequences through sampling and does not rely on function approximation or gradient-based learning. This makes it a suitable reference method and allows it to scale better than many classical mathematical optimization techniques. However, due to its sampling-based nature, CEM cannot provide formal guarantees of global optimality, even in relatively small environments.

The comparison with CEM is conducted in two stages. First, we consider a minimal toy environment consisting of a single quay wall. In this setting, the problem dimensionality is sufficiently small to allow CEM to, when run with a very large number of samples and iterations, be highly likely to identify the global optimum. To further increase confidence, CEM was executed multiple times using different random seeds, always converging to the same solution. While absolute optimality cannot be proven mathematically, these results make it very unlikely that the obtained solution is suboptimal. This setup therefore enables a direct validation of whether the DRL agent is capable of learning an optimal, or near-optimal, policy.

In this smallest environment, both CEM and the DRL agent converge to the same solution, achieving an identical total return of -2.1 M€. Moreover, the action sequences selected by both methods are identical, as shown in Figures 17 and 18. While CEM reaches this solution in approximately four minutes and the DRL agent requires around six minutes of training, the equivalence of the results strongly suggests that both methods have identified the optimal policy for this problem.

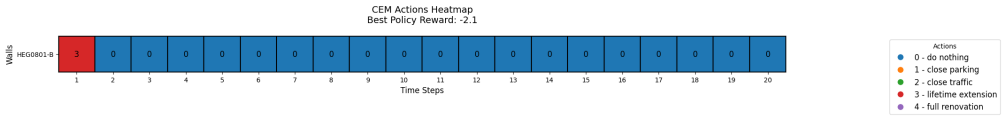

Figure 17: Optimal action sequence obtained with the CEM (single quay wall).

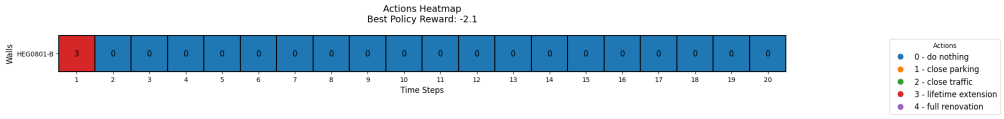

Figure 18: Optimal action sequence obtained with the DRL agent (single quay wall).

This result provides strong evidence that the developed DRL framework is capable of finding optimal solutions in low-dimensional settings.

In a second step, the same comparison is extended to environments of increasing size. Here, the goal is not to establish optimality, but to empirically demonstrate the curse of

dimensionality of mathematical optimization algorithms (despite the clear advantage of CEM against other mathematical optimization algorithms when it comes to scalability). Table 10 summarizes the results for different environment sizes.

Environment size	Method	Total reward	Computation time
1 quay wall	**DRL**	**-2.1 M €**	**≈ 6 min**
	CEM	**-2.1 M €**	≈ 4 min
7 quay walls	**DRL**	**-18.8 M €**	**≈ 27 min**
	CEM	-19.1 M €	≈ 11 h, 22 min
16 quay walls (3 groups)	**DRL**	**-29.4 M €**	**≈ 4 h, 7 min**
	CEM	-33.2 M €	≈ 35 h,1 min
32 quay walls (11 groups)	**DRL**	**-75.4 M €**	**≈ 3 h, 2 min***
	CEM	-94.2 M €	≈ 81 h, 33 min

* Simulation ran using warm start with curriculum learning

Table 10: Comparison of DRL and CEM performance across different environment sizes.

In Table 10 we can see how, as the number of components increases, the computational cost of CEM grows rapidly, while the DRL approach remains tractable. For the 16 component environment used throughout this section, it was possible to tune the CEM hyperparameters to obtain a reasonably good solution—substantially better than the rule-based benchmarks introduced earlier. However, achieving this solution required several days of simulation, whereas training the DRL agent required only a few hours. Moreover, the final performance achieved by the DRL agent is superior to that obtained by CEM.

A direct comparison between the training dynamics of both methods is shown in Figure 19.

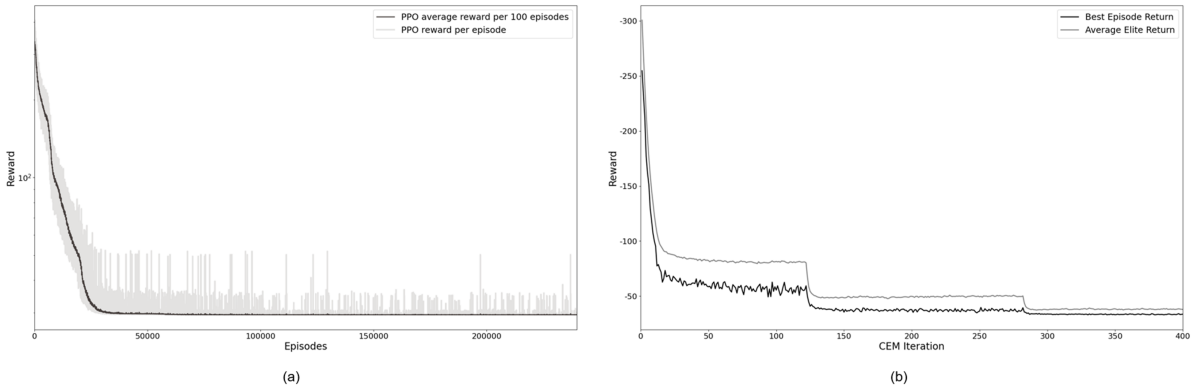

Figure 19: Training convergence of PPO (a) and CEM (b) for the 16-component environment.

As the environment size increases further, this trend becomes even more pronounced, as illustrated by the 32–quay-wall setting (Subsystem 2) reported in Table 10. These results highlight the significantly better scalability of the DRL approach compared to CEM.

Beyond quantitative performance, a qualitative analysis of the actions selected by both methods provides additional insights. Figure 20 compares the action heatmaps produced by CEM and the DRL agent for the 16–quay-wall subsystem.

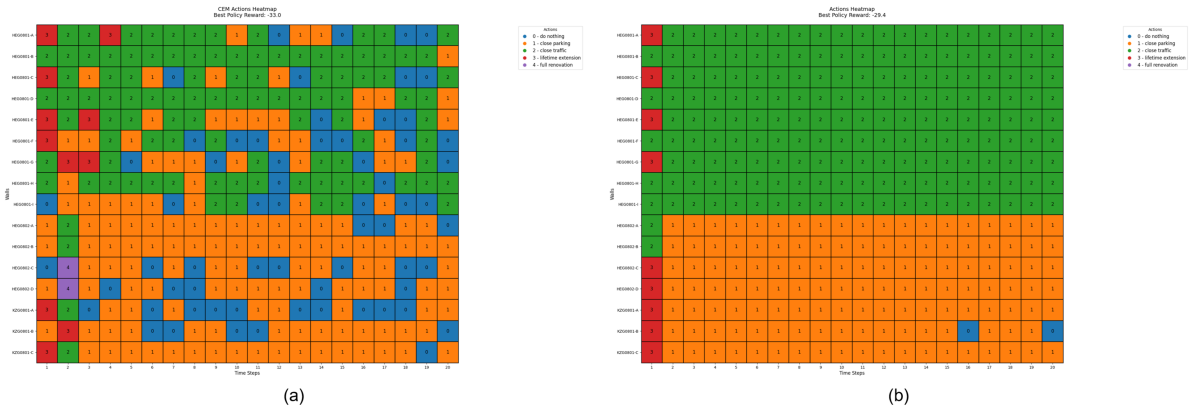

Figure 20: Action heatmap comparison for the 16–component subsystem: CEM (a) and DRL (b).

The actions selected by CEM appear irregular and lack a clear structural pattern, making them difficult to interpret from a domain perspective. In contrast, the DRL agent exhibits a coherent strategy. For example, when any quay wall within a group is closed for traffic, it is generally optimal to close the entire group, since city-level costs are incurred regardless, and closing the full group helps limit further degradation. While some inconsistencies are visible in the CEM solution, the DRL agent largely adheres to this logic. As environment size increases, differences between CEM and DRL action maps become more pronounced. For a more detailed view of this phenomenon, action maps for all environment sizes for both methods are provided in Appendix B.

7.2.1 Detailed Analysis of DRL Decisions in Subsystem 1

To better understand the policies learned by the DRL agent, we perform a detailed analysis of its decisions in the chosen 16–quay-wall-sections subsystem.

The DRL agent demonstrates a clear temporal structure in its decisions (Figure 20b). All maintenance actions involving direct costs—such as lifetime extensions and full renovations—are concentrated in the first year of simulation. In subsequent years, the agent focuses primarily on preserving network condition through traffic or parking closures. This behaviour is illustrated in Figure 21, which shows the yearly breakdown of different cost components.

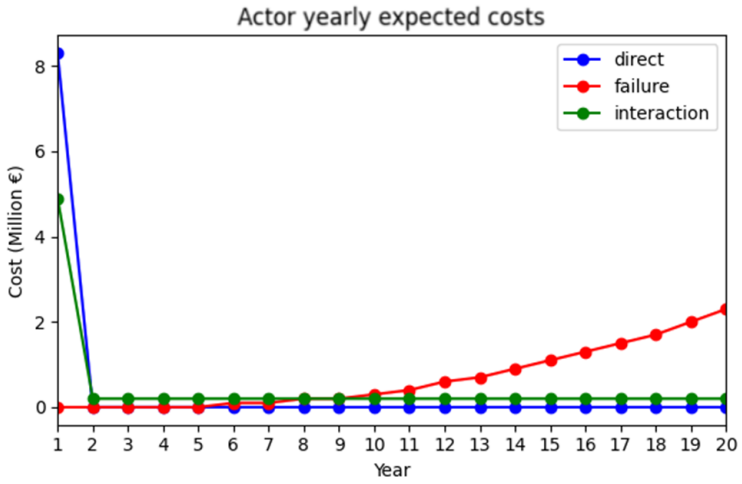

Figure 21: Yearly cost breakdown for the Subsystem 1 environment.

Direct maintenance costs occur only in the first year and drop to zero thereafter. City costs peak in the first year—when all quay-wall groups are closed for traffic—reaching approximately 4.9 M€. In subsequent years, when only one group is closed for traffic and the others are closed for parking, city costs decrease substantially to around 0.2 M€. Interestingly, the agent chooses to close traffic for group 1, while only closing parking for groups 2 and 3. This decision reflects the underlying city cost structure of the environment: closing traffic for group 1 alone is relatively cheap, whereas closing traffic for groups 2 and 3 simultaneously would result in very high costs (see Table C.1 in Appendix C, which details city costs for all possible closure scenarios).

A more detailed inspection of the agent's decisions further reveals that maintenance actions are strongly correlated with the initial condition of the quay walls. By comparing the action heatmap in Figure 20(b) with the initial state distribution shown in Figure 22, it can be observed that the agent consistently applies corrective maintenance to components starting in worse condition (dark orange), while selecting only preventive actions for components in better initial states (light orange or green).

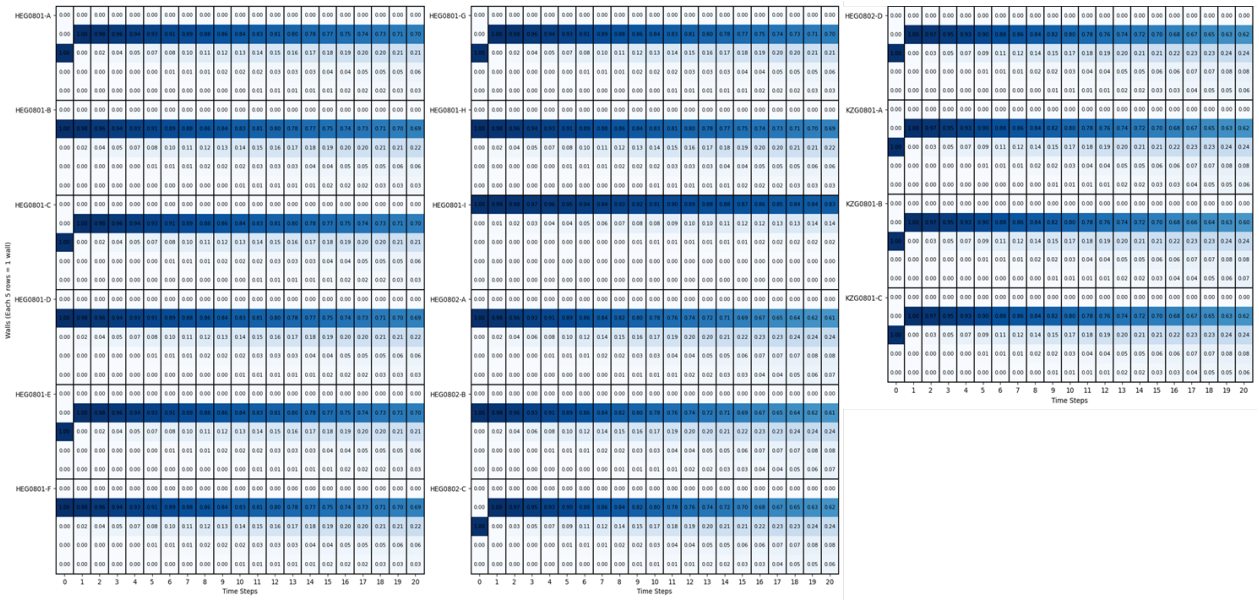

Figure 22: Initial state distribution of quay walls in the Subsystem 1 environment.

Overall, analyzing the different costs in the environment, the actions taken by the agent appear very adequate and close to optimal. The only inconsistency can be observed for section KZG0801-B in years 16 and 20, where the agent chooses to do nothing even though the entire group is closed for parking. This does not make sense, as vehicles would not be able to park in that section anyway, but in the model it continues to degrade as if it were fully exposed. This indicates that the agent did not find the absolute optimal action in this case. However, looking more closely at the description of this section (Table A.1 in Appendix A), we see that this is the shortest quay-wall segment in the environment (only 4 m), which results in very low failure costs. Consequently, these two suboptimal decisions only add a negligible cost (around 0.16 M€) and have minimal impact on the final reward obtained. Apart from this isolated case, no other incoherent decisions are apparent from visual inspection, suggesting that the learned policy is very close to the global optimum.

Finally, to help visualize the DRL agent's decision-making, Figure 23 presents a detailed view of a single quay-wall segment, showing its state evolution and the actions applied each year.

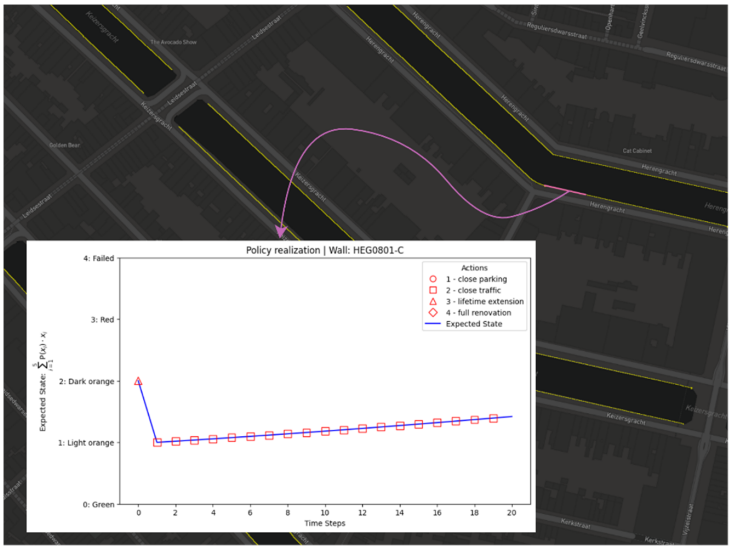

Figure 23: Evolution of wall HEG0801-C across the simulation horizon.

Figures 24 to 26 provide a broader overview, displaying the evolution of every component in each quay-wall group.

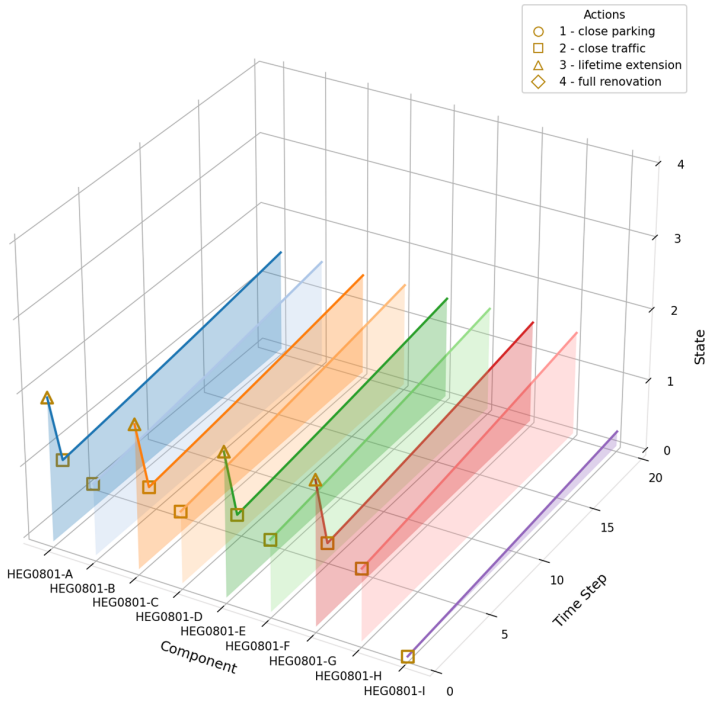

Figure 24: Evolution of every wall in group 1.

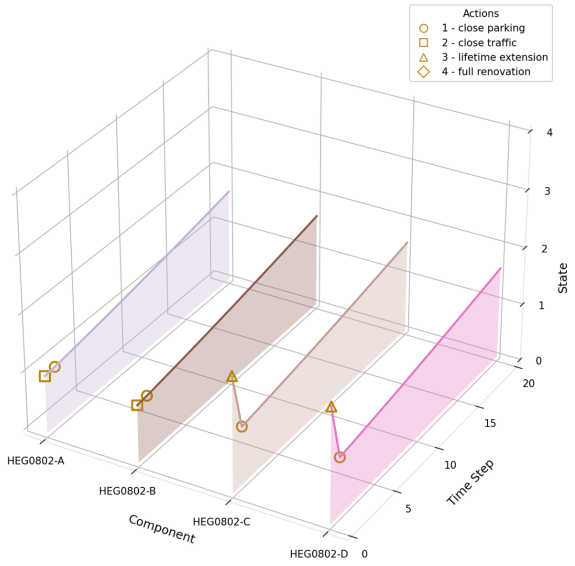

Figure 25: Evolution of every wall in group 2.

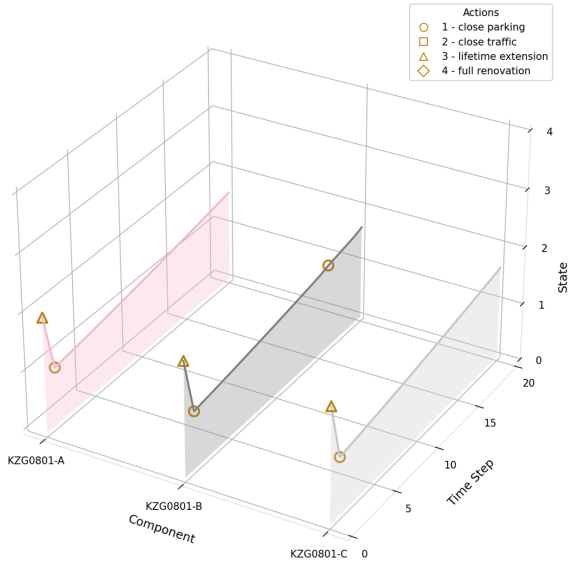

Figure 26: Evolution of every wall in group 3.

While this preservation strategy is mathematically optimal from a purely cost-minimization perspective, it highlights a practical limitation of the unconstrained model: the quay walls remain under some form of traffic or parking restriction for almost the entire planning horizon. In reality, maintaining consecutive, decades-long closures across major urban arteries is socially and politically infeasible due to the sustained disruption it causes to daily city life.

To prevent this unrealistic behavior, two primary approaches could be considered. The first is to introduce a hard constraint—for instance, utilizing action masking to enforce mandatory fully open periods (i.e., "do nothing" actions) after a certain number of consecutive closure years. Alternatively, this issue could be addressed without explicit constraints by dynamically adjusting the environment's cost structure. This would involve applying a time-dependent penalty to societal disruption, where the city cost of a traffic or parking closure scales non-linearly with each consecutive year the restriction is maintained, thereby naturally disincentivizing the agent from causing prolonged hindrances.

Although these operational mechanisms were evaluated at the beginning of this study, they were ultimately not implemented due to time limitations. Instead, prioritization was given to constraints (such as strict budget caps and failure limits) that have broader, immediate applicability across a wider range of general infrastructure maintenance problems. Nevertheless, restricting consecutive closures remains a highly relevant and necessary avenue for future research, as discussed further in Chapter 9.

7.3 Sensitivity Analysis: Effect of Modified Failure Cost Dynamics

Before introducing explicit constraints, we first analyze how changes in the environment dynamics alone influence the learned policy. This experiment serves two purposes. First, it further validates that the DRL agent is capable of adapting its strategy to different problem formulations beyond changes in environment size. Second, it illustrates how a decision maker can indirectly steer system behavior by adjusting cost structures, without imposing formal constraints on the policy space.

To this end, we consider a modified environment in which failure costs are multiplied by a factor of five. This modification represents a more risk-averse planning perspective, in which failures are treated as significantly more critical and are therefore strongly discouraged. Importantly, this is not a constraint in the sense of explicitly restricting the agent's action space. Instead, it alters the environment dynamics and the incentives faced by the agent. As such, it can be interpreted as a soft, easily tunable policy lever available to decision makers for shaping behavior without enforcing hard feasibility restrictions.

Under this modified cost structure, several interesting behavioral changes can be observed in Figure 27. As in the baseline setting, the agent still prioritizes performing major maintenance actions at the beginning of the planning horizon. However, to further reduce failure-related costs, the agent adopts a more conservative intervention strategy. For quay walls starting in poor condition (dark orange), the agent applies two consecutive lifetime extension actions, while components starting in light orange receive a single lifetime extension. For the quay wall initially in good condition (green), only preventive actions are selected. As a result, the policy effectively prioritizes bringing all components into a healthy state first and subsequently focuses on preserving that condition over time, which is reflected in the state evolution shown later.

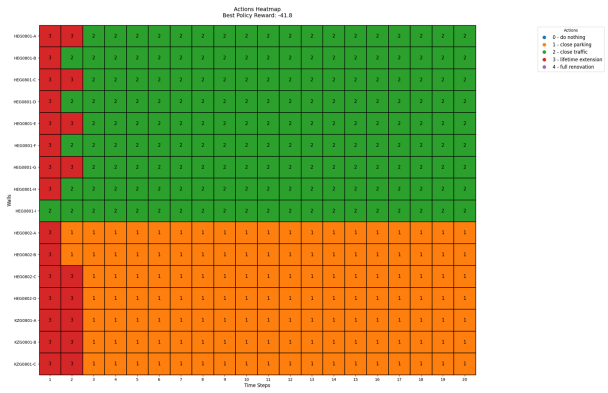

Figure 27: Action heatmap for the modified environment (5 × failure costs).

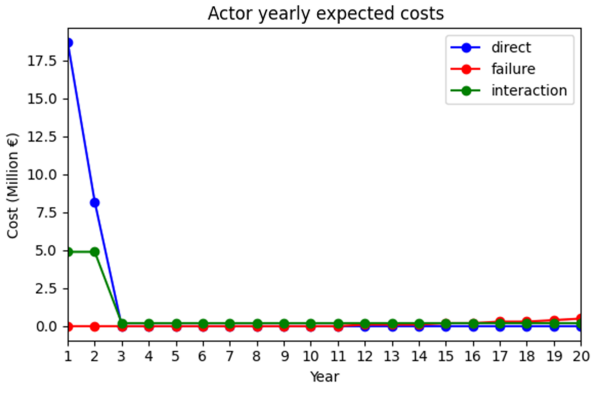

Figure 28: Cost breakdown for the modified environment (5 × failure costs).

A particularly noteworthy observation is the agent's preference for applying two consecutive lifetime extension actions instead of performing a single full renovation. While

both options incur approximately the same direct maintenance cost—since the cost of full renovation is roughly twice that of lifetime extension (Table A.1, Appendix A), they differ substantially in their dynamic effects. Performing two consecutive lifetime extensions implies closing the entire network for traffic for two years in a row, which increases city-level costs by approximately 4.7 M€(Figure 28). However, this strategy prevents degradation entirely during both intervention years, thereby significantly reducing expected failure costs. Because failure costs are heavily penalized in this scenario, this trade-off becomes favorable.

This behavior contrasts with the policies observed in later experiments, where the agent typically prefers a single full renovation to minimize city costs, even at the expense of slightly higher failure probabilities. The present results therefore highlight how modifying environment dynamics alone, without introducing explicit constraints, can lead to qualitatively different policies.

Finally, the resulting state trajectories confirm this interpretation. As shown in Figure 29, the agent rapidly drives all quay walls toward the green state and subsequently maintains them there, effectively minimizing the probability of failure throughout the horizon.

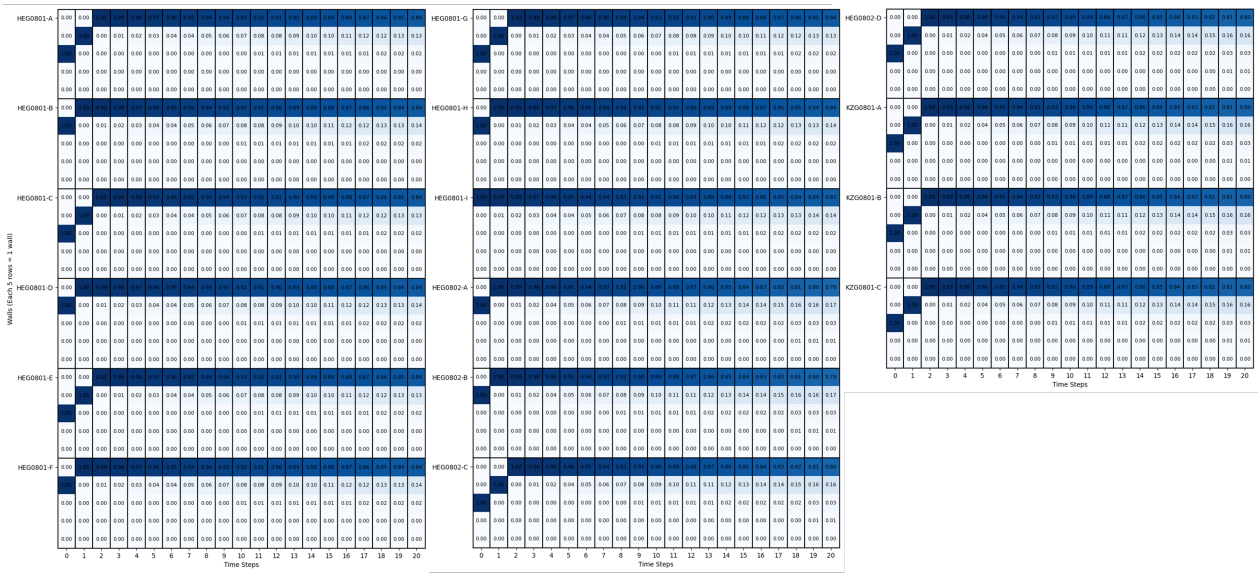

Figure 29: State heatmap for the modified environment (5 × failure costs).

Overall, this experiment demonstrates that adjusting cost weights provides a simple and flexible mechanism for influencing system behavior. While such an approach lacks the guarantees provided by explicit constraints, it can serve as a practical first step for decision makers who wish to discourage certain outcomes (such as failures), without increasing the complexity of the optimization problem. This sensitivity analysis also provides useful context for the constrained results presented in the following section.

7.4 Results for Constrained Optimization

In this section, we analyze the performance of the proposed DRL-based maintenance planning framework when explicit constraints are introduced. Building on the unconstrained results, the objective here is to assess whether the agent can (i) correctly enforce the imposed constraints, (ii) converge to a stable and feasible policy, and (iii) do so while maintaining reasonable performance in terms of total cost. Each subsection focuses on a specific constraint-handling mechanism introduced earlier and evaluates its practical impact on both the learned policy and the resulting system behavior.

7.4.1 Maximum Failure Probability Constraint ($P_{\mathbf{fail}} \leq 0.05$)

The first constraint considered imposes an upper bound of 0.05 on the allowable probability of failure for each individual quay wall. As shown by the unconstrained solution (Figure 22), this requirement is violated for several components under the optimal unconstrained policy. In particular, quay walls HEG0802-A, HEG0802-B, HEG0802-C, HEG0802-D, KZG0801-A, KZG0801-B, and KZG0801-C exhibit failure probabilities exceeding the prescribed threshold.

After introducing this constraint and retraining the agent, the model converges to a feasible solution with a total reward of -32.2 M€. A comparison between the resulting action policy (Figure 30) and the unconstrained case, shows that all quay walls that previously violated the constraint are now managed differently. Overall, the agent adopts a more conservative maintenance strategy. In particular, all violating quay walls are restored to a green state by year 2 through the application of stronger early interventions. Sections HEG0802-A and HEG0802-B shift from the "close traffic" action to "lifetime extension", while sections HEG0802-C, HEG0802-D, KZG0801-A, KZG0801-B, and KZG0801-C transition from "lifetime extension" to "full renovation".

Following these early corrective actions, the agent maintains the same structural pattern as in the unconstrained case by applying the same conservation measures for the remainder of the planning horizon. Consequently, city-level costs remain unchanged, while direct maintenance costs increase substantially and failure-related costs decrease accordingly, as illustrated in Figure 31.

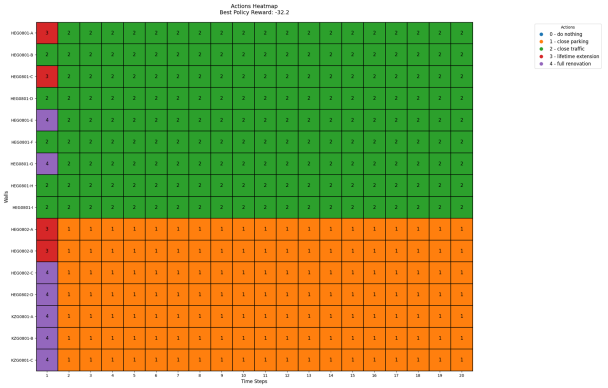

Figure 30: Actions heatmap with $P_{\text{fail}} \leq 0.05$.

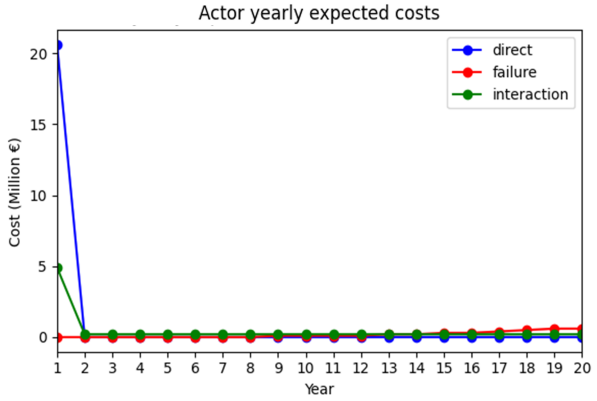

Figure 31: Yearly cost breakdown with $P_{\text{fail}} \leq 0.05$.

Finally, Figure 32 confirms the correct enforcement of the constraint. For all components and throughout the entire planning horizon, the probability of being in the failure state remains below the specified threshold of 0.05, including for those quay walls that exceeded this limit in the unconstrained case.

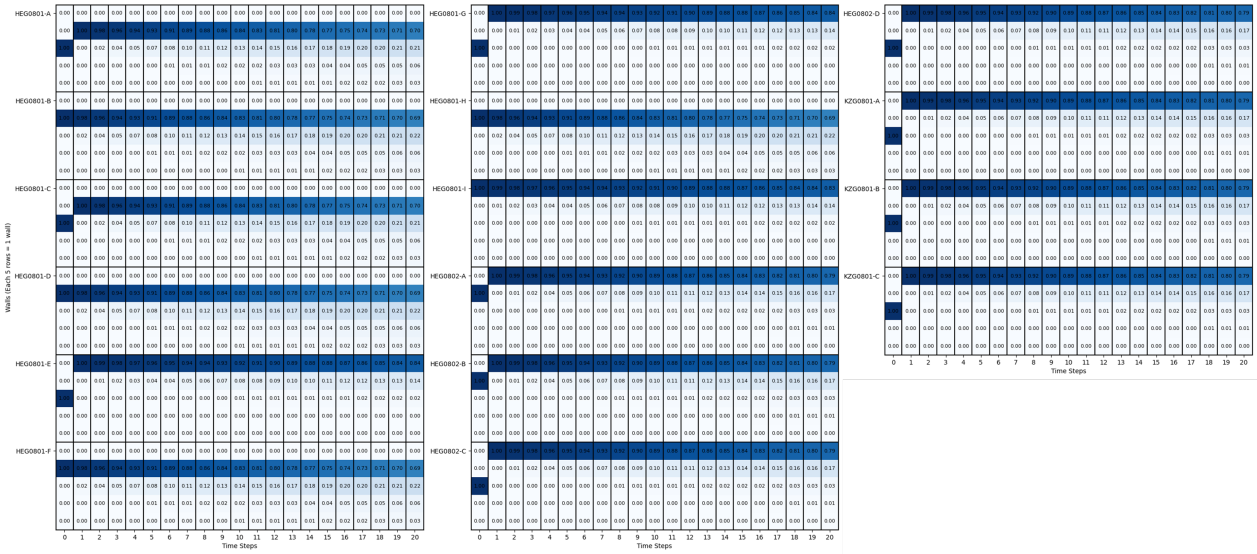

Figure 32: States heatmap with $P_{\text{fail}} \leq 0.05$.

7.4.2 Maximum Failure Cost Constraint ($\leq$ 0.8 M€ per year)

This constraint limits the total failure-related costs incurred in each year to a maximum of 0.8 M€. In the unconstrained case, yearly failure costs reached values of around 2.2 M€, indicating that failures were economically tolerated when preventive interventions were comparatively expensive. Imposing this bound therefore reflects a more conservative plan-

ning perspective, in which large failure-related expenditures are explicitly discouraged at the system level.

After introducing the constraint and retraining the agent, the model converges to a feasible solution with a total reward of -32.3 M€. The resulting action policy (Figure 33) shows that the agent continues to concentrate major maintenance actions in the first years of the planning horizon, consistent with the behavior observed in the unconstrained case.

A closer inspection of individual quay walls reveals that components responsible for higher failure costs in the unconstrained solution, such as KZG0801-C and HEG0802-B, which are among the longest sections in the environment, are managed more conservatively. For these quay walls, stronger early corrective actions are applied, while preventive actions in the later years of the horizon remain unchanged. This behavior can be explained by the high city-level costs associated with closing quay walls in groups 2 and 3, which would increase city costs by approximately 4.7 M€ per year and are therefore avoided by the agent.

As shown in Figure 34, city costs remain unchanged, while more conservative maintenance decisions lead to higher direct costs and significantly lower failure costs. Overall, the agent successfully keeps yearly failure costs below the imposed limit of 0.8 M€, confirming that the constraint is satisfied.

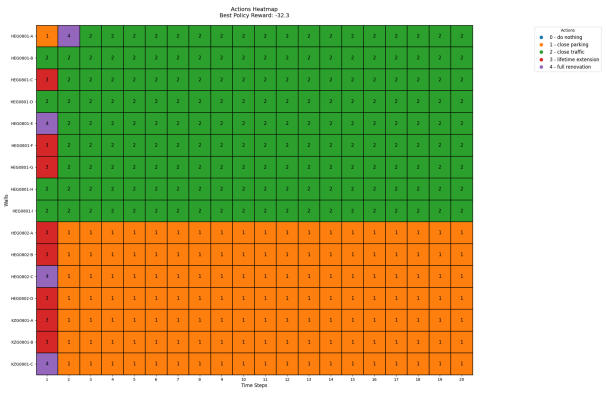

Figure 33: Action heatmap under the maximum failure cost constraint ($\leq$ 0.8 M€ per year).

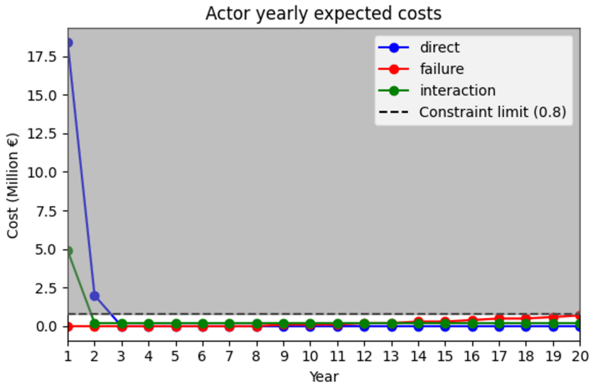

Figure 34: Yearly cost breakdown under the maximum failure cost constraint ($\leq$ 0.8 M€ per year).

7.4.3 Yearly Budget Constraint without Money Transfer ($\leq$ 2.5 M€ per year)

This constraint limits the maximum yearly expenditure on direct maintenance actions to 2.5 M€, with no possibility of transferring unused budget between years. After introducing this constraint and retraining the agent, the training converges to a feasible policy with a total reward of -50.1 M€, indicating a substantial degradation in performance compared to both the unconstrained and previously constrained cases.

An inspection of the resulting action policy (Figure 35) shows that maintenance actions are largely applied to the same quay walls as in the unconstrained case, but with notable timing and selection differences. In particular, actions associated with quay walls in group 1 are systematically delayed by one to two years. This behavior is consistent with the preventive effect of the "close traffic" action in group 1, which slows down deterioration and allows the agent to postpone interventions without immediately incurring large failure costs.

Two additional deviations are worth highlighting. First, for quay wall HEG0801-E, the agent selects "full renovation" instead of "lifetime extension". This choice is economically justified: delaying a lifetime extension from year 1 to year 3 would lead to higher cumulative failure costs over the horizon, making full renovation the more cost-effective option under the tight annual budget. Second, and more critically, no maintenance action is applied to quay wall KZG0801-C. This quay wall is among the longest in the subsystem and therefore associated with very high failure costs (Table A.1, Appendix A). However, all feasible maintenance actions for this component exceed the annual budget limit of 2.5 M€, effectively making any intervention infeasible. As a result, failure costs increase substantially in this case (Figure 36), driving the overall cost upward. Although closing traffic for group 3 could have reduced failure costs, the agent avoids this option due to the prohibitively high city-level costs it would induce.

Figure 36 confirms that the constraint is strictly enforced, as direct maintenance costs never exceed the annual limit of 2.5 M€.

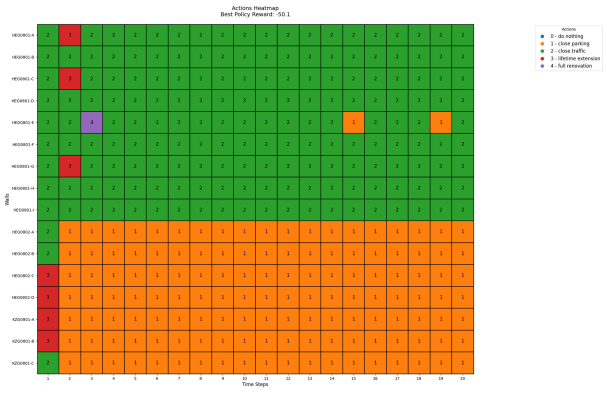

Figure 35: Action heatmap under the yearly budget constraint without transfer ($\leq$ 2.5 M€).

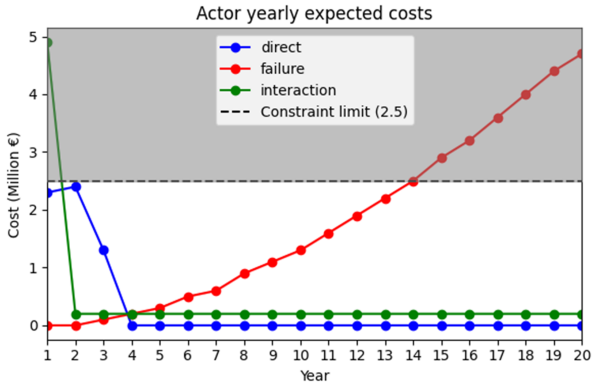

Figure 36: Yearly cost breakdown under the yearly budget constraint without transfer ($\leq$ 2.5 M€).

7.4.4 Yearly Budget Constraint with Money Transfer ($\leq$ 2.5 M€ per year)

In this variant, the same annual budget limit of 2.5 M€ is imposed, but unused budget can be carried over between years. Under this more flexible formulation, the agent converges to a significantly improved solution with a total reward of -32.0 M€, highlighting the importance of intertemporal budget flexibility.

The resulting action policy (Figure 37) closely matches the unconstrained solution in terms of which quay walls are eventually repaired, with only minor deviations in timing. As in the previous case, maintenance actions for group 1 quay walls are generally delayed, reflecting their slower deterioration dynamics. However, a key difference emerges for quay wall KZG0801-C. Unlike the no-transfer case, the agent now chooses to repair this component, recognizing that leaving it unaddressed would result in disproportionately high failure costs.

To enable this repair, the agent strategically accumulates budget by reducing maintenance actions in year 2 and performing no maintenance actions in year 3. This allows sufficient funds to be available for a full renovation of KZG0801-C, which is extremely costly due to its length (-5.6 M€, Table A.1, Appendix A). As a consequence, city costs increase noticeably in year 4 (by approximately 1 M€), since both groups 1 and 3 are closed to traffic during that year. The agent also prioritizes repairing KZG0801-C before quay walls HEG0801-E and HEG0801-G, because their associated failure costs are considerably lower.

Overall, this strategy leads to a substantial reduction in failure costs compared to the no-transfer case, while still respecting the yearly budget limit in every year, as shown in Figure 38.

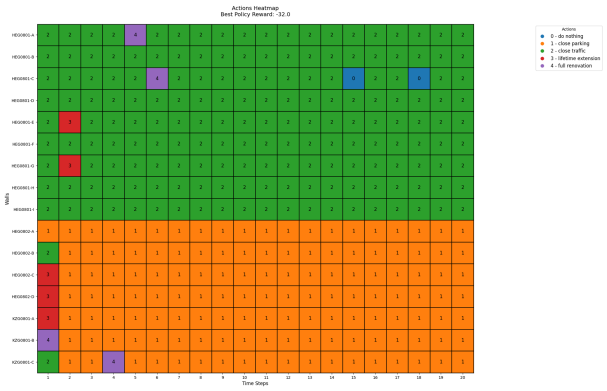

Figure 37: Action heatmap under the yearly budget constraint with transfer ($\leq$ 2.5 M€).

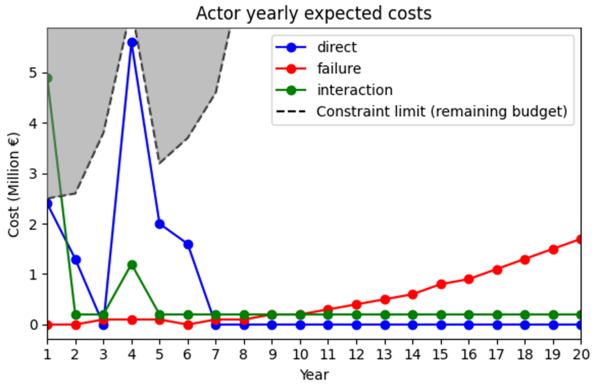

Figure 38: Yearly cost breakdown under the yearly budget constraint with transfer ($\leq$ 2.5 M€).

7.4.5 Budget Constraint over the Full Planning Horizon ($\leq$ 6 M€)

The final constraint limits the total direct maintenance expenditure over the entire planning horizon to 6 M€, without imposing any restriction on yearly spending. Under this global budget constraint, the agent converges to a feasible policy with a total reward of -40.7 M€.

The resulting action map (Figure 39) shows that, consistent with the unconstrained case, all maintenance actions are concentrated in the first year in order to minimize city-level costs. However, due to the limited total budget, only a subset of quay walls can be repaired. A

closer examination of Table A.1 (Appendix A) reveals that the agent makes economically sound trade-offs when allocating the available budget. Specifically, it prioritizes quay walls with the highest failure-cost-to-direct-cost ratio. As a result, all eligible quay walls from group 1 are selected, as they offer better cost-effectiveness than those in group 2. With the remaining budget, the agent repairs the longest quay wall in group 2, which is associated with the highest failure costs.

This selection strategy leads to a noticeable increase in failure costs relative to the unconstrained case, while successfully keeping total direct costs within the 6 M€ limit. As expected, city costs remain unchanged, since all actions are executed in the first year (Figure 40).

Despite the overall strong performance of the agent's action selection, a minor inconsistency can be observed for quay wall HEG0801-I. In this case, the agent repeatedly chooses the "do nothing" action even though the entire group is already closed to traffic. Under these conditions, selecting the "close traffic" action would have resulted in lower failure costs. Nevertheless, the impact of this suboptimal choice on the total cost remains quite small.

Figure 40 confirms that the overall budget constraint is satisfied.

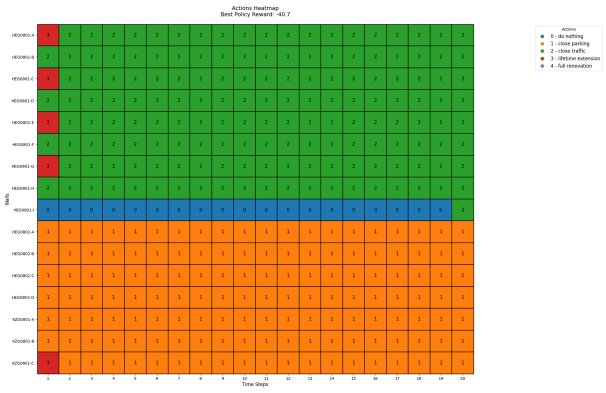

Figure 39: Action heatmap under the total budget constraint ($\leq$ 6 M€ over the full horizon).

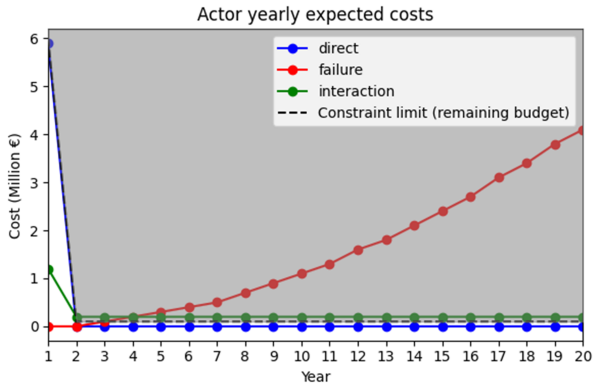

Figure 40: Yearly cost breakdown under the total budget constraint ($\leq$ 6 M€ over the full horizon).

7.4.6 Maximum City Costs ($\leq$ 4.5 M€)

The final constraint limits the total city-level costs over the full planning horizon. In the unconstrained problem, city costs reach approximately 8.0 M€, driven primarily by early traffic closures. Imposing an explicit upper bound on city costs therefore represents a strong system-level constraint that directly conflicts with the agent's tendency to concentrate maintenance actions in the first year.

As described in Section 6.8, this constraint is implemented using a PID-Lagrangian formulation. Since Lagrangian relaxation enforces constraints asymptotically and effectively

treats them as soft constraints, the target limit was set to 4.3 M€ rather than 4.5 M€ in order to provide a small safety margin and avoid systematic overshoot at convergence.

After extensive hyperparameter tuning, the PID controller successfully stabilizes the Lagrange multiplier, allowing the training process to converge to a feasible solution. Figure 41 shows the evolution of the multiplier, which settles at a value of approximately $\lambda = 0.89$. At this operating point, the agent converges to a stable policy with a total reward of -39.3 M€, while city costs are kept at 4.3 M€, satisfying the imposed constraint (Figures 42 and 43).

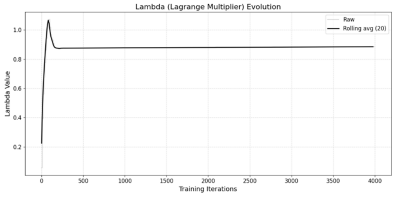

Figure 41: Evolution of the Lagrange multiplier under the maximum city cost constraint.

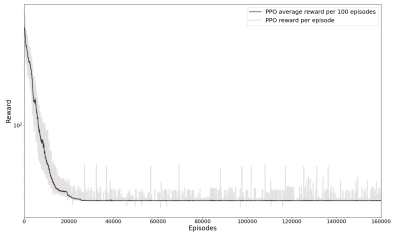

Figure 42: Training reward convergence under the maximum city cost constraint.

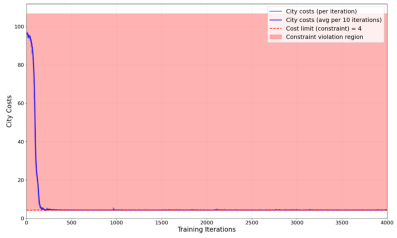

Figure 43: City cost evolution during training under the maximum city cost constraint.

Figure 42 illustrates the convergence of the training reward, while Figure 43 shows the corresponding evolution of city costs during training. As the multiplier stabilizes, city costs converge toward the target limit, confirming that the PID-Lagrangian mechanism effectively regulates the constraint without inducing persistent oscillations.

Qualitatively, an inspection of the learned action policy (Figure 44) reveals a clear structural shift compared to the unconstrained case. Specifically, the agent avoids closing group 2 to traffic in the first year, which is the main contributor to city-level costs. By postponing this intervention, the agent successfully reduces city costs to 4.3 M€, even though it is at the expense of higher failure-related costs (Figure 45).

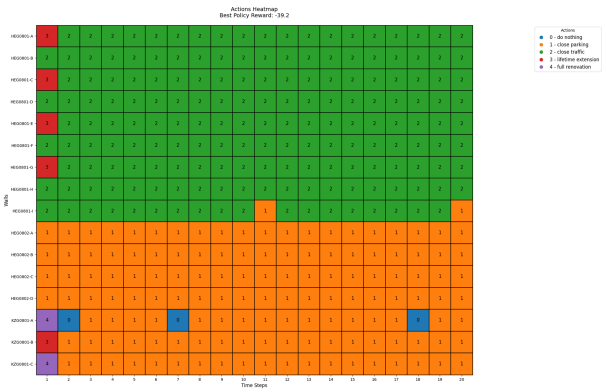

Figure 44: Action heatmap under the city cost constraint ($\leq$ 4.5 M€ over the full horizon).

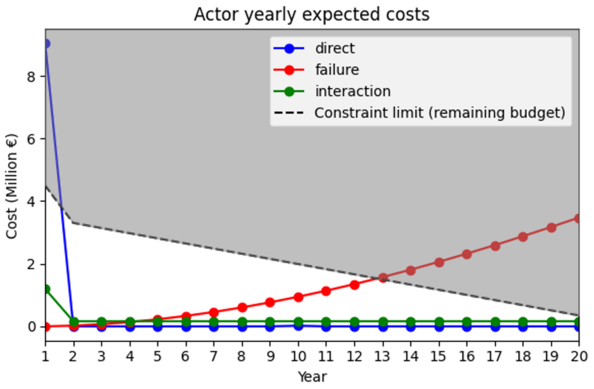

Figure 45: Yearly cost breakdown under the city cost constraint ($\leq$ 4.5 M€ over the full horizon).

Some minor inconsistencies can still be observed. For example, for quay wall KZG0801-A, the agent occasionally selects the "do nothing" action in steps 2, 7, and 18, even though the entire group is already closed to traffic. This behavior may also explain why the agent chooses "full renovation" instead of "lifetime extension" in the first step for this quay wall. Nevertheless, these effects remain local and do not compromise overall constraint satisfaction.

Overall, this experiment demonstrates that PID-Lagrangian relaxation can successfully enforce a challenging global cost constraint, even when it directly conflicts with the agent's natural cost-minimizing behavior. While performance in terms of total reward degrades relative to the unconstrained case, the agent converges to a stable and interpretable policy that respects the imposed city cost limit.

7.5 Effect of Warm-Start Training via Curriculum Learning

In this final part of the results section, we investigate the effect of warm-start training using curriculum learning. While all previous experiments focused on the quality and feasibility of the learned policies, the objective here is to evaluate training efficiency and scalability. In particular, we analyze whether initializing the agent with weights obtained from a related, simpler problem can (i) accelerate convergence, (ii) improve final performance, and (iii) remain robust when constraints or environment size change. Two complementary use cases are considered: warm starting to facilitate constrained optimization, and warm starting to scale the problem to larger environments.

7.5.1 Warm Start for Constrained Optimization

In a first experiment, curriculum learning is used to assess the effect of warm starting when introducing constraints in a fixed environment. The agent is initially trained on the same environment without constraints, and is then warm-started using the resulting unconstrained policy. The target task introduces a maximum failure probability constraint $P_{\text{fail}} \leq 0.05$, implemented via action masking.

The warm start is thus obtained by transferring the weights and biases of the unconstrained policy to the constrained setting, rather than training the constrained agent from scratch. All other hyperparameters are kept identical.

Figure 46 compares the learning curves of an agent trained from scratch (blue) and an agent warm-started using curriculum learning (red). The results show a clear acceleration in convergence: the warm-started agent reaches a stable and feasible solution significantly faster, while achieving a final reward comparable to the baseline (-33.4 M€ versus -33.2 M€). In both cases, the learned policies satisfy the imposed failure probability constraint across all quay walls and throughout the entire planning horizon.

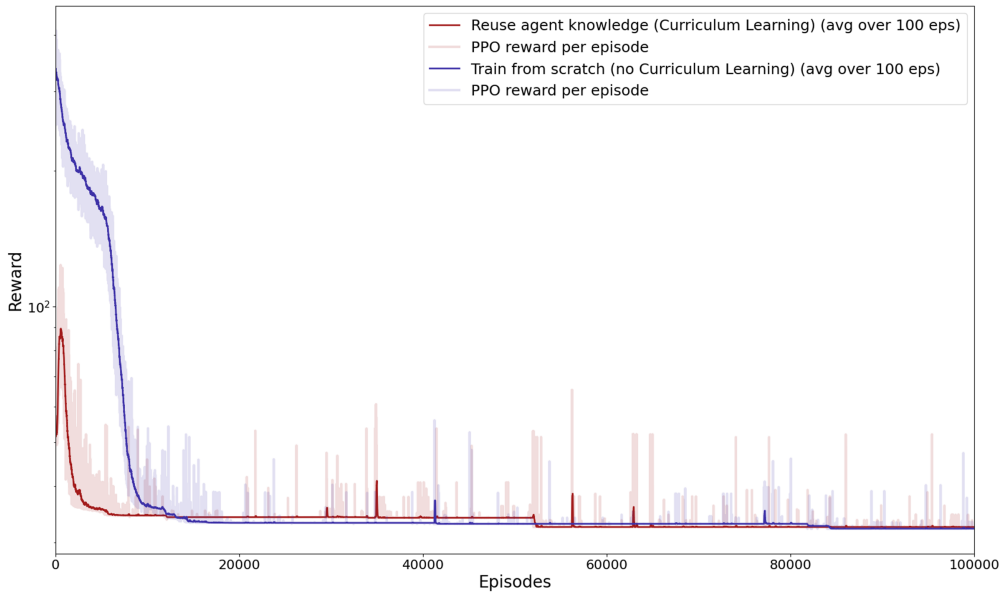

Figure 46: Training comparison of the DRL agent without (blue) and with (red) curriculum learning.

A practical challenge arises when warm starting from the unconstrained policy. Because the pretrained agent already exhibits low entropy, exploration is severely reduced at the beginning of constrained training. In our case, this led to the agent becoming trapped in a local optimum induced by the interaction between the pretrained policy and the action-masking constraint. To address this issue, the entropy coefficient was temporarily increased by a factor of 100 during the first 20 training iterations. This deliberate boost in exploration

produces the initial peak visible in the red learning curve in Figure 46 and allows the agent to escape the local optimum and discover a better feasible policy that continues to respect the imposed constraint.

Once this exploration phase concludes, the entropy coefficient is reset to its nominal value, and training proceeds normally. It should be noted that this behavior is closely tied to the type of constraint used, in this case action masking, and may differ for other constraint-handling mechanisms.

For the relatively small environment considered here, the benefits of warm starting are primarily reflected in faster convergence rather than improved final performance. The absence of a clear reward improvement in this setting suggests that, even when initialized randomly, the agent is able to reliably discover near-optimal feasible policies. This experiment therefore mainly demonstrates that warm starting does not degrade optimality in small-scale problems.

In larger and more complex environments, however, a reward improvement is expected. As the state and action spaces grow, randomly initialized agents are more likely to struggle with exploration and may converge to suboptimal feasible solutions. In such cases, initializing the constrained agent with a pretrained unconstrained policy can provide a more informative starting point, increasing the likelihood of reaching higher-quality solutions in addition to improving sample efficiency. From this perspective, the present results should be interpreted as a proof of feasibility in a small environment, while the full performance benefits of curriculum learning are expected to become more pronounced as problem scale increases.

At the same time, these results highlight an important caveat: warm starting is not universally beneficial. If the unconstrained and constrained optima are far apart, or if exploration is not handled carefully, curriculum learning can cause the agent to become trapped in a local optimum induced by the pretrained policy and the imposed constraints, thereby hindering optimization rather than helping it. Training from scratch, while slower, remains unbiased and more robust in such cases.

7.5.2 Warm Start for Scaling to Larger Environments

In a second experiment, curriculum learning is used to facilitate scalability. Rather than warm starting from an unconstrained policy, the agent is initialized using a constrained policy trained on a smaller environment. Specifically, the warm start is obtained from Subsystem 1, which consists of 16 QW sections. The constraint considered is again the maximum failure probability constraint $P_{\text{fail}} \leq 0.05$, but the environment size is subsequently increased.

The target environment corresponds to Subsystem 2, which contains 32 QW sections and explicitly embeds Subsystem 1 as a subset, ensuring structural continuity between the pretraining and target tasks. This nested structure allows the pretrained policy to be meaningfully transferred to the larger environment.

Figure 47 compares the training dynamics of an agent trained from scratch (blue) with

an agent warm-started using curriculum learning (red). The difference is substantial. The warm-started agent converges to a stable and feasible solution in approximately 5 hours, whereas training from scratch requires around 41 hours. In addition to faster convergence, curriculum learning also leads to superior final performance, with a higher total reward (-75.4 M€) than the scratch-trained agent (which converges to -81.8 M€). In both cases, the final policies satisfy the imposed failure probability constraint for all components over the full planning horizon.

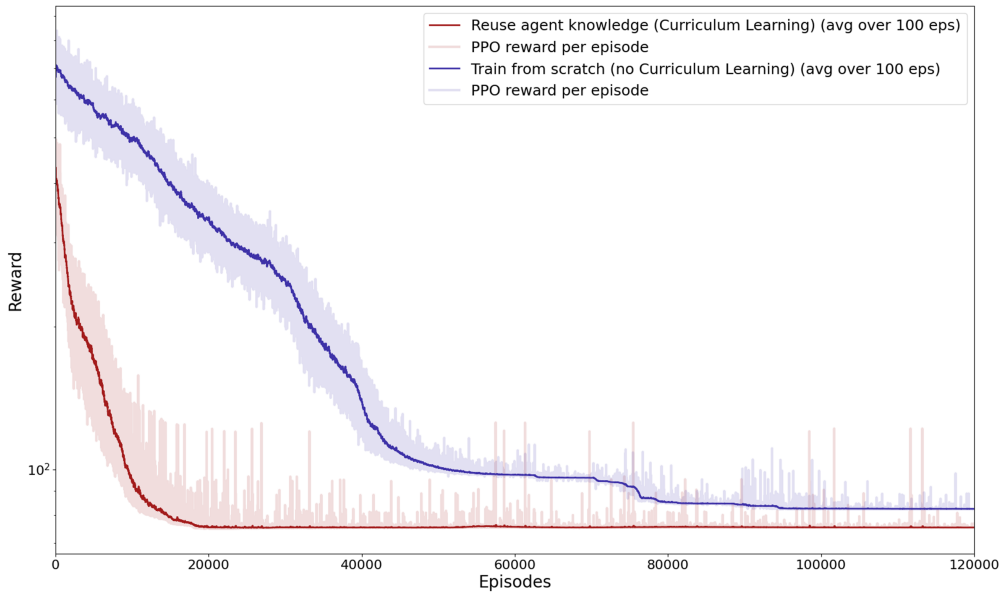

Figure 47: Training comparison for environment scaling without (blue) and with (red) curriculum learning.

These results demonstrate that warm starting is particularly effective when scaling the problem size, provided that structural similarities exist between the smaller and larger environments. The pretrained policy offers a strong inductive bias that guides exploration toward meaningful regions of the policy space, significantly reducing training time and improving solution quality, while still ensuring strict constraint satisfaction.

Overall, this experiment highlights curriculum learning as a powerful enabler for scaling constrained DRL-based infrastructure planning problems. While careful tuning is still required to avoid premature convergence or loss of exploration, warm-start strategies offer a practical pathway toward applying the proposed framework to realistically sized urban systems.

8 Discussion and Limitations

This thesis has developed and evaluated a multi-agent DRL framework for the strategic maintenance planning of Amsterdam's historic quay walls. By integrating high-fidelity city simulations with advanced constraint-handling mechanisms, this research bridges the gap between theoretical optimization and the operational realities of urban asset management.

This chapter synthesizes the experimental findings, evaluates the scientific and managerial relevance of the proposed framework, and acknowledges the limitations of the current study.

8.1 Performance and Optimality

A primary finding of this work is that the proposed DCMAC-PPO agent is capable of converging to optimal or near-optimal solutions across a variety of environment scales.

- **Superiority Over Current Practice:** Quantitative analysis reveals that the DRL agent achieves a total return three times higher than existing rule-based and reactive strategies. While current methods are often myopic or purely condition-based, the DRL agent exploits the underlying stochastic dynamics and non-linear city cost structures to find significantly more cost-effective policies.
- **Benchmarking and Scalability:** When validated against the CEM in a low-dimensional "toy" environment, the DRL agent identified the exact same optimal action sequences. Crucially, as the problem scale increased, the DRL agent maintained its performance and tractability, whereas the CEM suffered from the "curse of dimensionality", requiring days of computation to reach solutions that were still inferior to those produced by the DRL agent in a few hours.
- **Logical Coherence:** Qualitative inspection of the action heatmaps confirms that the agent's decisions are highly coherent. For instance, the agent learned to synchronize interventions within the same quay-wall group to minimize the "fixed" city costs of road closures for both parking and traffic. The rare, minor inconsistencies observed resulted in negligible reward differences, further suggesting that the agent has reached a policy very close to the global optimum.

8.2 Robustness of Constraint-Handling Mechanisms

The integration of real-world operational constraints was a central pillar of this research. The results demonstrate that the framework can successfully enforce both "hard" and "soft" constraints, making the model a realistic tool for asset managers.

- **Hard Constraints via Action Masking:** The combination of State Augmentation and Action Masking proved to be a highly effective and easily tunable method for satisfying strict requirements. Whether managing individual failure probabilities or annual maintenance budgets, the masking mechanism ensured that the agent's policy support never included invalid actions.
- **Soft Constraints via PID-Lagrangian:** For global, non-linear constraints like maximum city disruption, the PID-Lagrangian approach effectively regulated the Lagrange multiplier to satisfy safety targets. However, from an implementation standpoint, this method was notably more sensitive and required more extensive hyperparameter tuning compared to action masking.
- **Flexibility and Realism:** The agent's ability to adapt to different constraint sets—such as switching between strict annual budgets and flexible inter-temporal budget transfers—highlights the framework's versatility. It allows decision-makers to "tweak" the weights of different societal impacts and observe how the optimal policy evolves to stay within defined safety, financial, or disruption boundaries.

8.3 The Power of Curriculum Learning

Curriculum learning emerged as a powerful enabler for both training stability and scalability in constrained environments.

- **Facilitating Constraints:** Warm-starting a constrained agent using an unconstrained policy reached similar reward levels faster than training from scratch. A key finding was the necessity of temporarily increasing the entropy coefficient during this transition; this allowed the agent to "unlearn" unconstrained behaviors and escape the local optima induced by new action masks.
- **Enabling Scalability:** When scaling to larger environments (e.g., from 16 to 32 quay walls), curriculum learning provided a massive reduction in training time—from 41 hours down to 5. Notably, the warm-started agent also achieved a better final reward than the scratch-trained agent, suggesting that the inductive bias from smaller environments helps the agent navigate the vast search spaces of larger urban networks.

8.4 Scientific and Managerial Relevance

As outlined in the introduction of this thesis, applying DRL to infrastructure planning is an emergent field that has historically been confined to simplified, abstract "toy problems" or models that manage constraints through basic, unreliable reward shaping. This research successfully addressed this academic gap by rooting the DRL framework in a high-dimensional, multi-agent real-world case using validated municipal data. By moving beyond naive penalty

systems and implementing methodologically grounded constraint-handling techniques (Action Masking and PID-Lagrangian relaxation), this work advances the frontier of multi-agent DRL coordination under strict global feasibility requirements.

From a Management of Technology (MOT) perspective, this thesis effectively demonstrated how advanced decision-support models can function as a strategic corporate resource. Asset owners face the continuous challenge of managing aging infrastructure under tight financial, safety, and societal pressures. By incorporating the US digital twin directly into the DRL pipeline, this methodology provides a data-driven tool capable of balancing these competing objectives. Even in scenarios where the generated policy might require minor manual adjustments, the framework supplies a high-quality, mathematically grounded starting point that is vastly more efficient than traditional, siloed planning methods. Because the architecture is modular, this approach can easily be extrapolated to other infrastructure asset domains—such as bridge networks or utility grids—worldwide.

8.5 Limitations

While the proposed DRL framework demonstrates strong potential, several limitations should be acknowledged:

First, the environment considered in this study represents only a section of Amsterdam's quay wall network. Although scalability tests indicate that the framework can handle larger environments, applying it to an entire city will require additional model refinement and computational resources. Nevertheless, for smaller towns or less complex networks, the current methodology is already close to practical applicability.

Second, certain modeling assumptions limit the realism of the current environment. The discount factor (0.99) may not fully align with real-world market interest rates, potentially affecting the relative weighting of near-term versus long-term costs. Maintenance costs were applied uniformly per meter across all quay walls, overlooking variations in accessibility, complexity, or intervention difficulty. Additionally, city-level costs were grouped into clusters of three quay walls, which may not fully capture interaction effects when multiple quay walls are simultaneously closed. This simplification could underestimate or overestimate cumulative disruption.

9 Conclusions and Future Work

9.1 Conclusions

This research set out to explore how deep reinforcement learning can be adapted to solve highly constrained, real-world infrastructure maintenance challenges. By systematically developing and validating the framework, this thesis provides clear answers to the sub-research questions that guided the investigation:

1. What types of real-world constraints and heuristics are most relevant to urban infrastructure maintenance, and how should they be treated?

The most critical constraints identified for urban infrastructure maintenance broadly span safety, financial, and societal domains. As demonstrated through the Amsterdam quay wall case study, safety requirements (e.g., maximum allowable failure probabilities) and financial limitations (e.g., strict annual or multi-year budgets) represent inviolable operational boundaries. These must be treated as *hard constraints* to ensure structural integrity and prevent budget overruns. Conversely, limitations regarding network-wide societal impacts—such as urban hindrance, traffic disruption, and environmental emissions—are better formulated as *soft constraints*. Treating these societal factors as soft constraints allows them to act as system-level targets that the agent must minimize and balance globally, providing the necessary flexibility to navigate complex, non-linear network interdependencies without rendering the optimization problem infeasible.

2. Which DRL constraint-handling methods are best suited for integrating these factors?

For hard, component-level, or purely additive constraints, *Action Masking* combined with *State Augmentation* proved to be the most effective and reliable method, completely preventing the agent from selecting mathematically or financially invalid actions. For soft, non-linear, and global constraints—such as city disruption where action masking is computationally intractable—*PID-Lagrangian Relaxation* using a dual-critic architecture emerged as the most suitable method to dynamically balance cost minimization with constraint adherence.

3. How can inter-agent constraints be effectively enforced in a Multi-agent DRL framework, and how can the scalability of the training process be facilitated for larger infrastructure networks?

Inter-agent coordination was successfully achieved through sequential evaluation and centralized state tracking. For additive constraints like shared budgets, action masking was applied dynamically, updating the remaining shared resources after each agent's decision. For global constraints like city costs, the PID-Lagrangian approach allowed multiple independent agents to learn cooperative strategies by subjecting them to a shared, network-level penalty parameter (λ) that regulated overall system disruption. Furthermore, to facilitate the scala-

bility of this multi-agent framework to larger networks, Curriculum Learning (warm-starting) proved to be highly effective. By transferring policy weights from smaller, previously solved subsystems, the constrained framework demonstrated rapid scalability to much larger environments, drastically cutting training times without sacrificing solution quality or constraint adherence.

4. How well does the proposed DRL framework perform in generating valid and efficient intervention plans?

The framework demonstrated exceptional performance, reliably converging to policies that balanced asset reliability with cost-effectiveness. The agent generated intervention plans that minimized overall network costs while strictly adhering to all imposed constraints, effectively resolving the complex trade-offs faced by municipal asset managers.

5. How does the constrained DRL method compare to baselines and unconstrained DRL?

The DRL agent achieved total returns approximately three times higher than existing rule-based and reactive baselines, demonstrating that current heuristic-driven maintenance strategies can be significantly improved through learning-based approaches. Furthermore, the proposed model matched the accuracy of classical mathematical optimization (CEM) on small-scale problems while effectively overcoming the curse of dimensionality on larger networks, improving the obtained reward and reducing computational time from days to mere hours. This stark contrast highlights the practical superiority of DRL in high-dimensional, real-world scenarios where traditional mathematical optimization methods become intractable. When compared to its unconstrained counterpart, the constrained DRL agent successfully adapted its underlying policy to guarantee operational feasibility while still obtaining highly competitive rewards.

The insights gained from these five sub-research questions collectively provide a definitive and successful resolution to the main research question guiding this study:

How can real-world constraints and domain-specific heuristics be effectively integrated into DRL models to generate practical, scalable, and cost-effective maintenance policies for complex infrastructure systems?

Ultimately, this research demonstrates that real-world constraints and heuristics can be seamlessly integrated into DRL models through a modular, multi-layered architecture. By translating structural limits and budgets into hard mathematical boundaries via state-augmented Action Masking, and by regulating global societal disruptions through PID-Lagrangian relaxation, the DRL agent is forced to optimize strictly within the bounds of operational reality. Furthermore, integrating external digital twins (such as the Urban Strategy tool) allows the model to capture realistic, non-linear environmental dynamics, while employing Curriculum Learning ensures that the training process remains scalable and computationally tractable. Together, these techniques successfully transform theoretical DRL

algorithms into practical, scalable, and cost-effective decision-support tools perfectly tailored for the complexities of modern urban infrastructure systems.

9.2 Future Research

Future work can build on this framework in several directions:

One avenue is improving the fidelity of the environment. This includes modeling structural degradation for individual quay wall sections, refining urban cost impacts, and addressing limitations in the current city costs grouping methodology to better capture spatial interactions.

Another important research direction is expanding the model to larger environments to evaluate whether the agent can maintain near-optimal performance in true city-scale networks. In this context, further exploration of advanced warm-start strategies for constrained optimization is highly recommended.

Action coordination within quay wall groups is another promising enhancement. For example, if one quay wall in a group is scheduled to close parking or restrict traffic, the same action could automatically apply to the other walls in the group. Hard-coding this through Action Masking could eliminate the minor synchronization mistakes observed in the agent's behavior and improve computational efficiency by shrinking the search space.

Another significant direction is incorporating explicit observation or inspection actions to transition the environment into an active POMDP. Currently, the framework takes actions based on the belief state of the different quay walls; however, in practice, physical inspections can be scheduled to determine the exact structural condition of a specific element before making a maintenance decision. Allowing the DRL agent to actively decide when to inspect would force it to learn the Value of Information (VoI)—balancing the immediate cost of gathering data against the long-term risk of unobserved structural failure. While this would increase the model's realism and applicability, it also introduces substantial training challenges, as dealing with delayed rewards and expanding belief spaces would heavily impact scalability and training time. In such a setting, the environment would need to be simulated in a fully stochastic manner.

Furthermore, as the environment scales to a city-wide level, exploring more decentralized multi-agent architectures, such as DDMAC, presents a highly relevant research avenue. While the current framework successfully coordinates actions and enforces shared constraints using a more centralized approach, transitioning to a more decentralized architecture could significantly enhance training scalability and reduce computational overhead. However, future studies must carefully evaluate how this decentralization affects the final results—specifically, whether decentralized agents can still effectively satisfy global constraints (such as shared multi-year budgets and network-wide city costs) and maintain the high level of near-optimal coordination achieved by the current model.

Finally, several additional constraints and heuristics could be integrated to increase realism and flexibility:

- **Spatial clustering of maintenance activities:** Grouping nearby interventions could reduce mobilization costs and streamline logistics.
- **Penalizing consecutive interventions:** Adding a penalty for repeated actions on the same quay wall could help distribute maintenance more evenly over time.
- **End-of-horizon deterioration:** Explicitly accounting for residual deterioration at the end of the planning horizon may influence the learned policy and improve long-term asset quality.
- **Prioritizing critical elements:** Designating specific quay walls as high-priority to ensure they remain in a green or minimally degraded state throughout the planning horizon.
- **"Do nothing" periods for specific elements or groups:** Enforcing minimum open periods for traffic or restricting the total number of simultaneous closures in a subsystem to guarantee baseline city accessibility.
- **Scheduling constraints:** Prioritizing maintenance during quieter periods to avoid high-demand tourist seasons or major city events.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] E. Diederichs, "Reinforcement learning-a technical introduction," *Journal of Autonomous Intelligence*, vol. 2, no. 2, p. 25, 2019.

[3] G. P. Tsinker, *Handbook of port and harbor engineering*. Boston, MA: Springer US, 1997.

[4] Y. Hou, X. Liang, J. Zhang, Q. Yang, A. Yang, and N. Wang, "Exploring the use of invalid action masking in reinforcement learning: A comparative study of on-policy and off-policy algorithms in real-time strategy games," *Applied Sciences*, vol. 13, no. 14, p. 8283, 2023.

[5] U. Waseem, "Pid controller & loops: A comprehensive guide to understanding and implementation," June 21 2023.

[6] M. Landers, "Pid lagrangian," 2025.

[7] OpenAI, "Proximal policy optimization." `https://spinningup.openai.com/en/latest/algorithms/ppo.html`, 2022.

[8] D. Luongo, G. Nicodemo, A. Venmans, M. Korff, L. Sartorelli, H. Maljaars, and D. Peduto, "The quay walls of amsterdam, netherlands: An approach for collapse risk mitigation at the municipal scale based on multisource monitoring and surveying data," *Journal of Geotechnical and Geoenvironmental Engineering*, vol. 152, 2024.

[9] Gemeente Amsterdam, "Voortgangsrapportage programma bruggen en kademuren: Inhoudelijke rapportage over periode januari 2023 – maart 2024, financiële rapportage over periode januari – december 2023." `https://openresearch.amsterdam/image/2024/6/3/voortganghsrapportage_januari_2023_maart_2024_bruggen_en_kademuren_mei_2024.pdf`, May 2024.

[10] R. E. Wildeman, R. Dekker, and A. C. J. M. Smit, "A dynamic policy for grouping maintenance activities," *European Journal of Operational Research*, vol. 99, no. 3, pp. 530–551, 1997.

[11] O. Kammouh, C. Fecarotti, and A. Marandi, "A scalable optimization approach to the intervention planning of complex interconnected infrastructures," *Reliability Engineering & System Safety*, vol. 250, p. 110281, 2024.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2 ed., 2018.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[14] C. P. Andriotis and K. G. Papakonstantinou, "Managing engineering systems with large state and action spaces through deep reinforcement learning," *Reliability Engineering & System Safety*, vol. 191, p. 106483, 2019. Preprint at arXiv:1811.02052.

[15] C. P. Andriotis and K. G. Papakonstantinou, "Deep reinforcement learning driven inspection and maintenance planning under incomplete information and constraints," *Reliability Engineering & System Safety*, vol. 212, p. 107551, 2021.

[16] L. W, C. H, B. J, K. R, A. Y, and W. E, "Building digital twins of cities using the inter model broker framework," *Future Generation Computer Systems*, no. 148, pp. 501–513, 2023.

[17] H. Dong, Z. Ding, and S. Zhang, *Deep reinforcement learning: fundamentals, research and applications*. Springer Nature, 2020.

[18] J. F. Andersen, A. R. Andersen, M. Kulahci, and B. F. Nielsen, "A numerical study of markov decision process algorithms for multi-component replacement problems," *European Journal of Operational Research*, vol. 292, no. 2, pp. 616–628, 2021.

[19] M. Compare, P. Marelli, P. Baraldi, and E. Zio, "A markov decision process framework for optimal operation of monitored multi-state systems," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 232, no. 6, pp. 677–689, 2018.

[20] R. Schöbi and E. N. Chatzi, "Maintenance planning using continuous-state partially observable markov decision processes and non-linear action models," *Structure and Infrastructure Engineering*, vol. 12, no. 8, pp. 977–994, 2016.

[21] R. B. Corotis, J. H. Ellis, and M. Jiang, "Modeling of risk-based inspection, maintenance and life-cycle cost with partially observable markov decision processes," *Structure and Infrastructure Engineering*, vol. 1, no. 1, pp. 75–84, 2005.

[22] K. T. Nguyen, P. Do, K. T. Huynh, C. Bérenguer, and A. Grall, "Joint optimization of monitoring quality and replacement decisions in condition-based maintenance," *Reliability Engineering & System Safety*, vol. 189, pp. 177–195, 2019.

[23] K. G. Papakonstantinou and M. Shinozuka, "Planning structural inspection and maintenance policies via dynamic programming and markov processes. part ii: Pomdp implementation," *Reliability Engineering & System Safety*, vol. 130, pp. 214–224, 2014.

[24] K. G. Papakonstantinou, C. P. Andriotis, and M. Shinozuka, "Pomdp and momdp solutions for structural life-cycle cost minimization under partial and mixed observability," *Structure and Infrastructure Engineering*, vol. 14, no. 7, pp. 869–882, 2018.

[25] C. P. Andriotis, K. G. Papakonstantinou, and E. N. Chatzi, "Value of structural health information in partially observable stochastic environments," *Structural Safety*, vol. 93, p. 102072, 2021.

[26] P. G. Morato, K. G. Papakonstantinou, C. P. Andriotis, J. S. Nielsen, and P. Rigo, "Optimal inspection and maintenance planning for deteriorating structural components through dynamic bayesian networks and markov decision processes," *Structural Safety*, vol. 94, p. 102140, 2022.

[27] P. G. Morato, J. S. Nielsen, A. Mai, and P. Rigo, "Pomdp based maintenance optimization of offshore wind substructures including monitoring," *Proceedings of the 13th International Conference on Applications of Statistics and Probability in Civil Engineering (ICASP13)*, 2019.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[30] GeeksforGeeks, "Actor-critic algorithm in reinforcement learning," 2024.

[31] J. G. de Gijt, *A history of quay walls, techniques, types, costs and future*. PhD thesis, Delft University of Technology, Delft, 2010.

[32] A. De Graauw, "Ancient port structures," *Méditerranée*, 2022.

[33] Amstelodamum, "De oorsprong van de amsterdamse kademuren," 2021.

[34] CUR Commissie 186, *Binnenstedelijke kademuren*. Rotterdam, Netherlands: SBRCURnet, 2013.

[35] Gemeente Amsterdam, "Kademuren: maatregelen en vernieuwen." `https://www.amsterdam.nl/projecten/kademuren/`, 2023. Accessed: 2023.

[36] L. Neijzing and R. Wesstein, "Amsterdamse risicobeoordeling kademuren - versie 1," report, Gemeente Amsterdam, Amsterdam, 2022.

[37] W. Lohman, H. Cornelissen, J. Borst, Y. Araghi, and E. Walraven, "Building digital twins of cities using the inter model broker framework," *Future Generation Computer Systems*, vol. 148, pp. 501–513, 2023.

[38] C. Lathourakis and A. Martínez, "Deep reinforcement learning for life-cycle maintenance planning of large infrastructure networks," in *Proceedings of the 36th European Safety and Reliability Conference (ESREL 2026)* (J. C. Matos, P. B. Lourenço, M. Beer, D. Oliveira, and E. Patelli, eds.), Research Publishing, 2026.

[39] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the International Conference on Machine Learning*, pp. 1889–1897, PMLR, 2015.

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[41] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang, "The 37 implementation details of proximal policy optimization," in *ICLR Blog Track*, 2022.

[42] S. Shaik, J. M. Smereka, and Y. Wang, "Generalized advantage estimation for distributional policy gradients," in *2025 American Control Conference (ACC)*, (Denver, CO, USA), pp. 3073–3078, 2025.

[43] L. Swaalf, D. M. A. Spruijtenburg, R. Sinha, H. Weijs, and P. d. Heer, "Dragon's den project planning maintenance," tech. rep., TNO, Delft, 2023. Unpublished.

[44] P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, no. 1, pp. 19–67, 2005.

[45] S. Mannor and R. Y. Rubinstein, "The cross-entropy method for fast policy search," in *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*, pp. 512–519, 2003.

[46] A. Beren, "Cem as inference." `https://www.beren.io/2023-03-30-CEM-as-inference/`, 2023.

[47] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*. Dordrecht: D. Reidel, 1987.

[48] O. Vinyals *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[49] C. Berner *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[50] D. Ye *et al.*, "Mastering complex control in moba games with deep reinforcement learning," in *AAAI Conference on Artificial Intelligence*, vol. 34, pp. 6672–6679, 2020.

[51] C.-Y. Tang, C.-H. Liu, W.-K. Chen, and S. D. You, "Implementing action mask in proximal policy optimization (ppo) algorithm," *ICT Express*, vol. 6, no. 3, pp. 200–203, 2020.

[52] E. Altman, *Constrained Markov Decision Processes*, vol. 7. CRC Press, 1999.

[53] A. Stooke, J. Achiam, and P. Abbeel, "Responsive safety in reinforcement learning by pid lagrangian methods," in *International Conference on Machine Learning*, pp. 9133–9143, PMLR, 2020.

[54] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," *arXiv preprint arXiv:1805.11074*, 2018.

[55] A. Ray, J. Achiam, and D. Amodei, "Benchmarking safe exploration in deep reinforcement learning," *arXiv preprint arXiv:1910.01708*, vol. 7, 2019.

[56] J. Ji, J. Zhou, B. Zhang, J. Dai, X. Pan, R. Sun, W. Huang, Y. Geng, M. Liu, and Y. Yang, "Omnisafe: An infrastructure for accelerating safe reinforcement learning research," *arXiv preprint arXiv:2305.09304*, 2023.

[57] "Safe policy optimization documentation," 2023. Accessed: 2026-01-11.

[58] K. J. Astrom and T. Hagglund, "Pid control," *IEEE Control Systems Magazine*, vol. 26, no. 1, pp. 6–23, 2006.

[59] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual conference on International conference on machine learning*, pp. 41–48, ACM, 2009.

Statement on the Use of Generative AI Tools

During the course of this master's thesis, large language models (LLMs) were used as a supporting tool in a limited and controlled manner. Their use was intended to assist the research process, not to replace critical thinking, domain expertise, or engagement with scientific literature.

First, LLMs were employed for textual refinement, specifically to improve grammar, readability, and clarity of drafts written by the author. This included rephrasing sentences, improving flow, and correcting linguistic errors. The underlying content, structure, and scientific arguments were always authored by the researcher.

Second, LLMs were used for brainstorming purposes, such as exploring alternative formulations, identifying potential angles for discussion, or generating high-level suggestions during early stages of ideation. These outputs served only as prompts to stimulate thinking and were selectively adopted, modified, or discarded based on the researcher's judgment.

Third, LLMs were occasionally used as an aid for conceptual understanding, for example to obtain preliminary explanations of unfamiliar concepts or methods. In all such cases, the information provided by the LLMs was treated as non-authoritative. Any conceptual understanding derived from these interactions was subsequently verified through peer-reviewed literature, textbooks, or other authoritative academic sources, which are cited accordingly in this thesis.

At no point were LLMs used to generate original scientific results, perform data analysis, or draw conclusions. All claims, interpretations, and findings presented in this thesis are based on the author's own work and on referenced academic sources. The use of LLMs was therefore supportive and editorial in nature, consistent with responsible research practices and academic integrity standards.

Appendix A. Quay Wall Data and Cost Parameters

quay_wall_id	group_id	length	direct cost (lifetime extension)	direct cost (full renovation)	failure cost
HEG0801-A	1	39.5	-0.99 M €	-1.98 M €	-3.29 M €
HEG0801-B	1	70	-1.75 M €	-3.50 M €	-3.50 M €
HEG0801-C	1	31.5	-0.79 M €	-1.58 M €	-2.63 M €
HEG0801-D	1	47.5	-1.19 M €	-2.38 M €	-3.17 M €
HEG0801-E	1	26	-0.65 M €	-1.30 M €	-2.17 M €
HEG0801-F	1	19	-0.48 M €	-0.95 M €	-1.58 M €
HEG0801-G	1	26	-0.65 M €	-1.30 M €	-2.17 M €
HEG0801-H	1	43	-1.08 M €	-2.15 M €	-2.15 M €
HEG0801-I	1	42	-1.05 M €	-2.10 M €	-2.10 M €
HEG0802-A	2	46	-1.15 M €	-2.30 M €	-2.30 M €
HEG0802-B	2	194	-4.85 M €	-9.70 M €	-9.70 M €
HEG0802-C	2	51	-1.28 M €	-2.55 M €	-2.55 M €
HEG0802-D	2	33	-0.83 M €	-1.65 M €	-2.20 M €
KZG0801-A	3	5.6	-0.14 M €	-0.28 M €	-0.37 M €
KZG0801-B	3	4	-0.10 M €	-0.20 M €	-0.20 M €
KZG0801-C	3	111.72	-2.79 M €	-5.59 M €	-7.45 M €
KZG0802-A	4	15	-0.38 M €	-0.75 M €	-1.25 M €
KZG0802-B	4	109.5	-2.74 M €	-5.48 M €	-7.30 M €
KZG0901-A	5	248.5	-6.21 M €	-12.43 M €	-16.57 M €
KZG0901-B	5	5.9	-0.15 M €	-0.30 M €	-0.49 M €
KZG0901-C	5	5.7	-0.14 M €	-0.28 M €	-0.47 M €
LEG0101-A	6	106	-2.65 M €	-5.30 M €	-5.30 M €
LEG0102-A	7	105	-2.63 M €	-5.25 M €	-7.00 M €
LEG0201-A	8	122	-3.05 M €	-6.10 M €	-6.10 M €
LEG0202-A	9	106	-2.65 M €	-5.30 M €	-7.07 M €
LEG0301-A	10	32.2	-0.81 M €	-1.61 M €	-1.61 M €
LEG0301-B	10	18.9	-0.47 M €	-0.95 M €	-0.95 M €
LEG0301-C	10	63.7	-1.59 M €	-3.19 M €	-3.19 M €
LEG0301-D	10	9.1	-0.23 M €	-0.46 M €	-0.76 M €
KZG0902-A	11	120	-3.00 M €	-6.00 M €	-8.00 M €
KZG0902-B	11	16	-0.40 M €	-0.80 M €	-1.07 M €
KZG0902-C	11	110	-2.75 M €	-5.50 M €	-7.33 M €
KZG1001-A	12	8.5	-0.21 M €	-0.43 M €	-0.43 M €
KZG1001-B	12	128	-3.20 M €	-6.40 M €	-6.40 M €
KZG1001-C	12	24	-0.60 M €	-1.20 M €	-1.20 M €
KZG1001-D	12	14.5	-0.36 M €	-0.73 M €	-0.73 M €
KZG1001-E	12	10	-0.25 M €	-0.50 M €	-0.67 M €
KZG1002-A	13	30	-0.75 M €	-1.50 M €	-2.50 M €
KZG1002-B	13	136	-3.40 M €	-6.80 M €	-11.33 M €
KZG1002-C	13	2.3	-0.06 M €	-0.12 M €	-0.19 M €
KZG1002-D	13	8.5	-0.21 M €	-0.43 M €	-0.71 M €
PRG0901-A	14	285	-7.13 M €	-14.25 M €	-14.25 M €
PRG0901-B	14	37	-0.93 M €	-1.85 M €	-1.85 M €
PRG0902-A	15	128	-3.20 M €	-6.40 M €	-6.40 M €
PRG0902-B	15	198	-4.95 M €	-9.90 M €	-9.90 M €
PRG1002-A	16	112	-2.80 M €	-5.60 M €	-5.60 M €
PRG1002-B	16	27	-0.68 M €	-1.35 M €	-1.35 M €
PRG1002-C	16	97	-2.43 M €	-4.85 M €	-4.85 M €
PRG1001-A	17	5	-0.13 M €	-0.25 M €	-0.42 M €
PRG1001-B	17	144	-3.60 M €	-7.20 M €	-9.60 M €
PRG1001-C	17	21	-0.53 M €	-1.05 M €	-1.40 M €
PRG1001-D	17	74	-1.85 M €	-3.70 M €	-4.93 M €
SGG0101-A	18	41	-1.03 M €	-2.05 M €	-2.05 M €
SGG0101-B	18	12	-0.30 M €	-0.60 M €	-0.60 M €
SGG0101-C	18	45	-1.13 M €	-2.25 M €	-2.25 M €
SGG0102-A	19	32	-0.80 M €	-1.60 M €	-2.13 M €
SGG0102-B	19	19	-0.48 M €	-0.95 M €	-0.95 M €
SGG0102-C	19	40	-1.00 M €	-2.00 M €	-2.00 M €
SIG0802-A	20	149	-3.73 M €	-7.45 M €	-9.93 M €
PRG0803-A	21	42	-1.05 M €	-2.10 M €	-2.10 M €
PRG0803-B	21	99	-2.48 M €	-4.95 M €	-4.95 M €
HEG0702-A	22	57.2	-1.43 M €	-2.86 M €	-3.81 M €
HEG0702-B	22	15.1	-0.38 M €	-0.76 M €	-1.01 M €
HEG0702-C	22	26.3	-0.66 M €	-1.31 M €	-2.19 M €
LEG0402-A	23	4.5	-0.11 M €	-0.23 M €	-0.15 M €
LEG0402-B	23	19.1	-0.48 M €	-0.96 M €	-0.64 M €
LYG0902-A	23	11.6	-0.29 M €	-0.58 M €	-0.77 M €
LYG1101-A	24	210	-5.25 M €	-10.50 M €	-14.00 M €
LYG1101-B	24	13.5	-0.34 M €	-0.68 M €	-0.90 M €
LYG1201-A	25	115	-2.88 M €	-5.75 M €	-7.67 M €

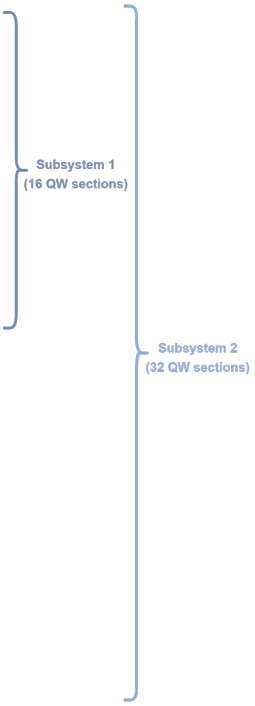

Table A.1: Optimal action sequence obtained with CEM for a single quay wall.

Appendix B. Action Maps for Different Environment Sizes

This appendix presents the action maps obtained with the Cross-Entropy Method (CEM) and the DRL agent (PPO) for all environment sizes considered in the validation experiments. The figures provide a visual comparison of the action sequences selected by both methods across increasing problem scales, supporting the qualitative discussion in Section 7 on policy structure, interpretability, and scalability. For each environment size, the CEM and DRL results are shown side by side to facilitate direct comparison.

B.1 Single Quay Wall

CEM

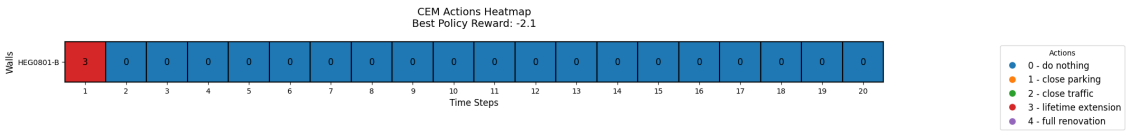

Figure B.1: Optimal action sequence obtained with CEM for a single quay wall.

DRL

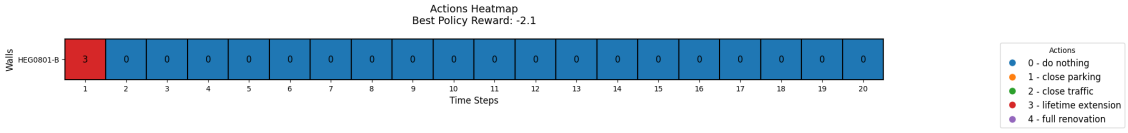

Figure B.2: Optimal action sequence obtained with the DRL agent for a single quay wall.

B.2 Seven Quay Walls

CEM

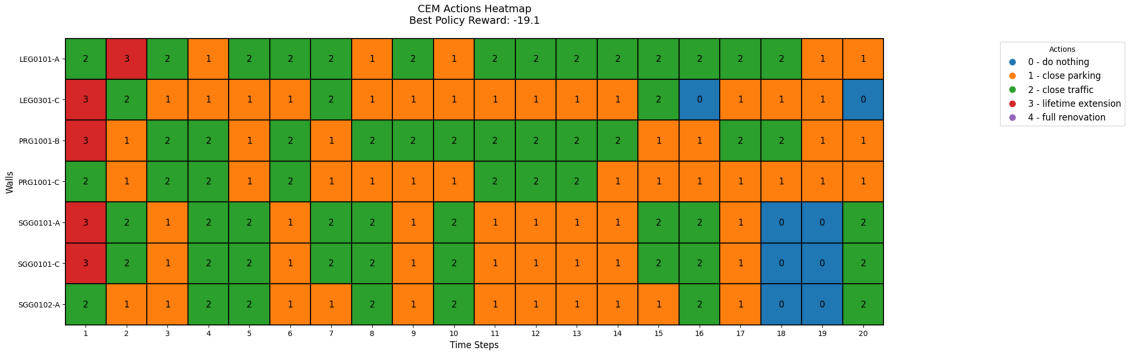

Figure B.3: Optimal action sequence obtained with CEM for seven quay walls.

DRL

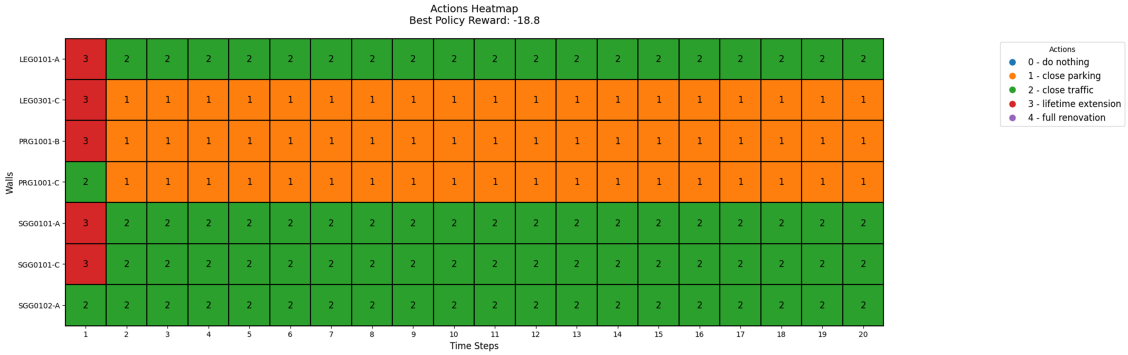

Figure B.4: Optimal action sequence obtained with the DRL agent for seven quay walls.

B.3 Sixteen Quay Walls (Subsystem 1)

CEM

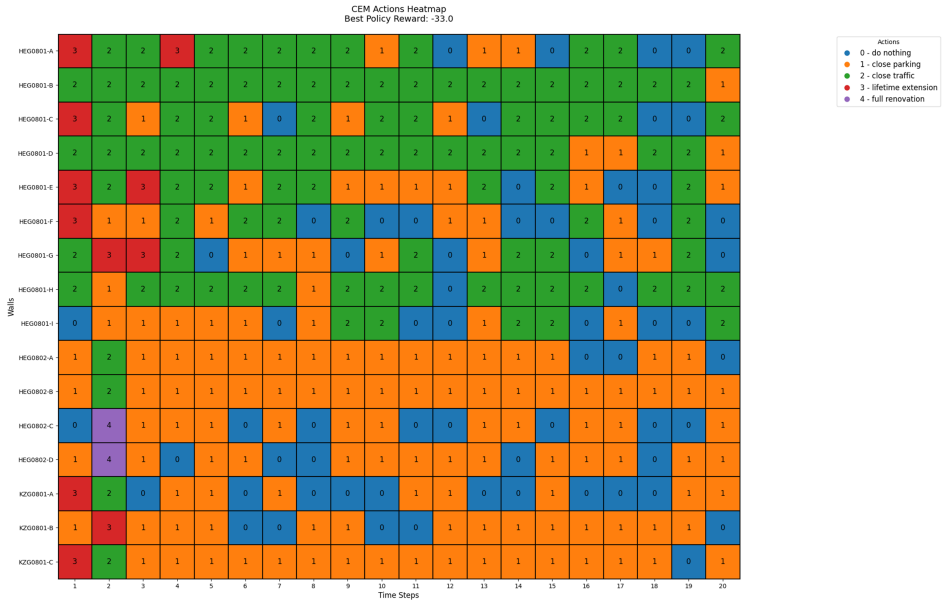

Figure B.5: Optimal action sequence obtained with CEM for sixteen quay walls.

DRL

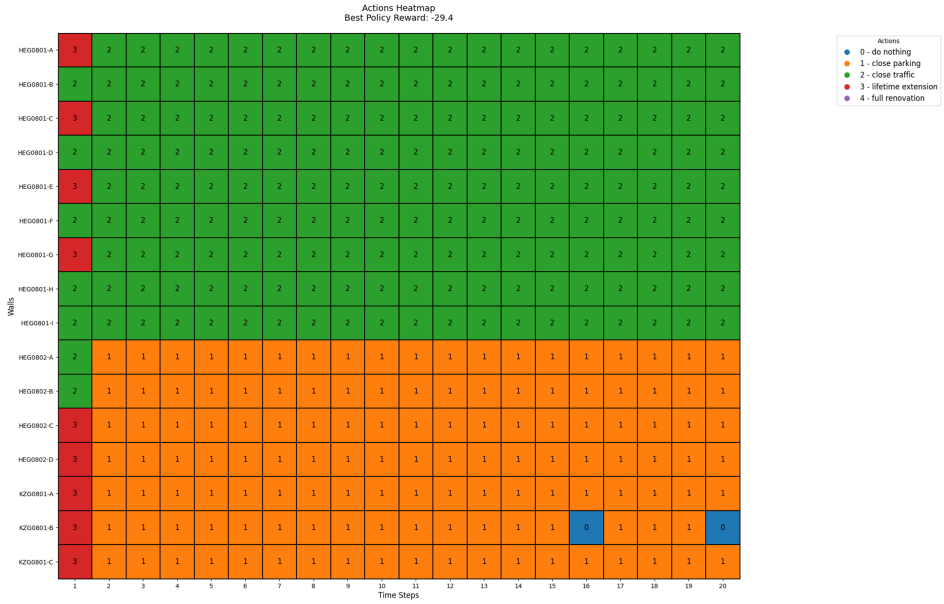

Figure B.6: Optimal action sequence obtained with the DRL agent for sixteen quay walls.

B.4 Thirty-Two Quay Walls (Subsystem 2)

CEM

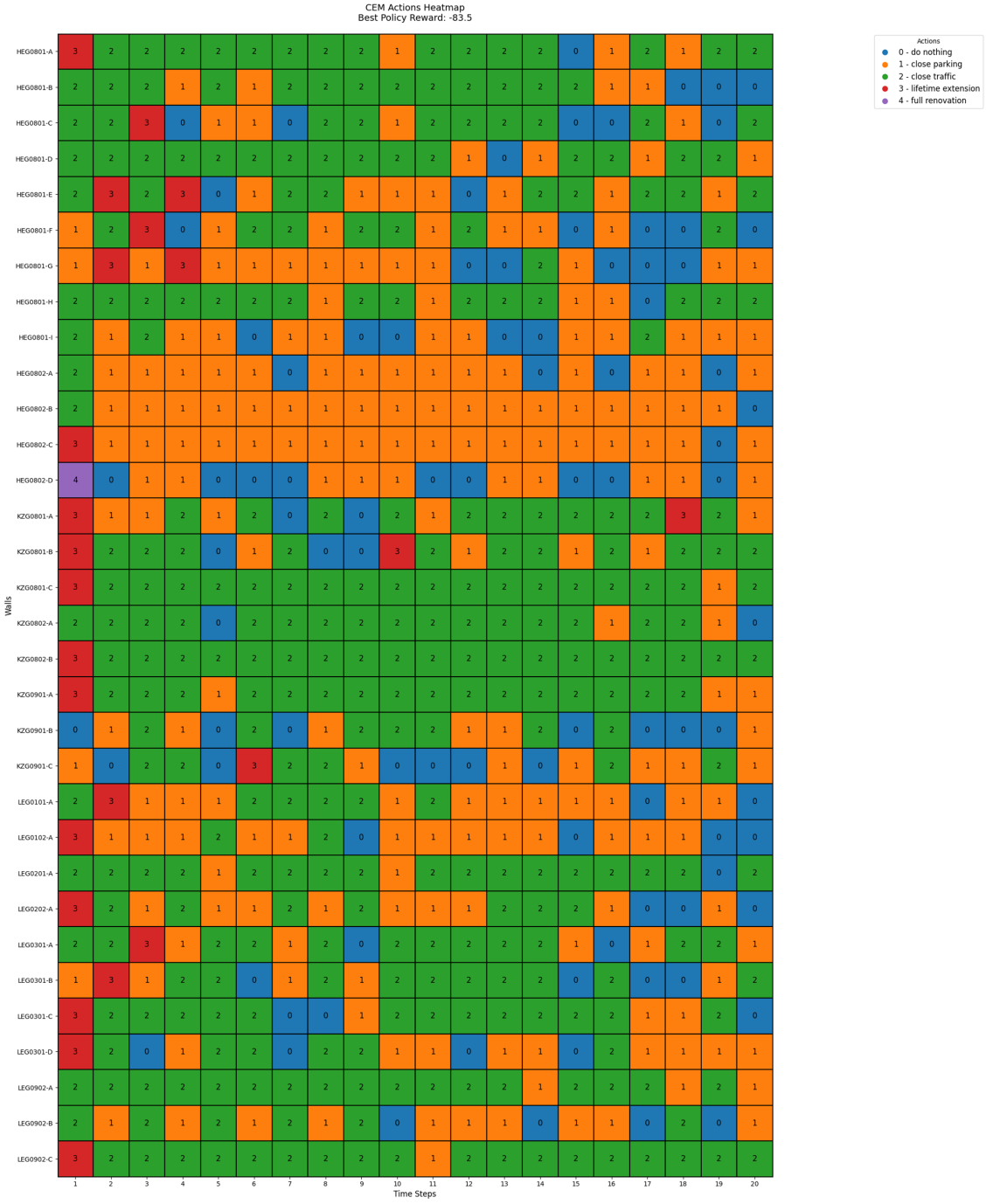

Figure B.7: Optimal action sequence obtained with CEM for thirty-two quay walls.

DRL

Figure B.8: Optimal action sequence obtained with the DRL agent for thirty-two quay walls.

Together, these figures illustrate how differences between CEM and DRL become increasingly pronounced as the environment size grows, with the DRL agent exhibiting more structured and interpretable action patterns in larger-scale settings.

Appendix C. Possible Combinations of City Costs in Subsystem 1

Action	Group_id_1	Group_id_2	Group_id_3	Cost
close parking	1	2	3	0.65 M €
close traffic	1	2	3	4.87 M €
close parking	1	2		0.22 M €
close traffic	1	2		5.44 M €
close parking	1			0.16 M €
close traffic	1			0.13 M €
close parking	1	3		0.57 M €
close traffic	1	3		0.97 M €
close parking	2			0.22 M €
close traffic	2			5.80 M €
close parking	2	3		0.04 M €
close traffic	2	3		5.61 M €
close parking	3			0.20 M €
close traffic	3			0.07 M €

Table C.1: City costs for each possible combination of group closures in Subsystem 1