Delft University of Technology

Deep multi-objective reinforcement learning for utility-based infrastructural maintenance optimization

van Remmerden, Jesse; Kenter, Maurice; Roijers, Diederik M.; Andriotis, Charalampos; Zhang, Yingqian; Bukhsh, Zaharah

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Deep multi-objective reinforcement learning for utility-based infrastructural maintenance optimization

Jesse van Remmerden[1] · Maurice Kenter[2] · Diederik M. Roijers[2,3] · Charalampos Andriotis[4] · Yingqian Zhang[1] · Zaharah Bukhsh[1]

Abstract

In this paper, we introduce multi-objective deep centralized multi-agent actor-critic (MO-DCMAC), a multi-objective reinforcement learning method for infrastructural maintenance optimization, an area traditionally dominated by single-objective reinforcement learning (RL) approaches. Previous single-objective RL methods combine multiple objectives, such as probability of collapse and cost, into a singular reward signal through reward-shaping. In contrast, MO-DCMAC can optimize a policy for multiple objectives directly, even when the utility function is nonlinear. We evaluated MO-DCMAC using two utility functions, which use probability of collapse and cost as input. The first utility function is the threshold utility, in which MO-DCMAC should minimize cost so that the probability of collapse is never above the threshold. The second is based on the failure mode, effects, and criticality analysis methodology used by asset managers to assess maintenance plans. We evaluated MO-DCMAC, with both utility functions, in multiple maintenance environments, including ones based on a case study of the historical quay walls of Amsterdam. The performance of MO-DCMAC was compared against multiple rule-based policies based on heuristics currently used for constructing maintenance plans. Our results demonstrate that MO-DCMAC outperforms traditional rule-based policies across various environments and utility functions.

Keywords Reinforcement learning · Multi-objective reinforcement learning · Maintenance · Infrastructure

1 Introduction

For any nation, a robust and functional infrastructure system is required for the efficient transportation of commercial goods, individuals, and essential services, such as clean water and electricity. This is evidenced by the distinct correlation between a country's gross domestic product (GDP) and the level of development of its infrastructure [26]. Given this direct relationship, it is vital to have a comprehensive maintenance strategy that ensures

✉ Jesse van Remmerden
j.v.remmerden@tue.nl

Maurice Kenter
m.kenter@amsterdam.nl

Diederik M. Roijers
d.roijers@amsterdam.nl

Charalampos Andriotis
C.Andriotis@tudelft.nl

Yingqian Zhang
yqzhang@tue.nl

Zaharah Bukhsh
z.bukhsh@tue.nl

[1] Information Systems IE&IS, Eindhoven University of Technology, De Zaale, 5600 MB Eindhoven, The Netherlands

[2] City of Amsterdam, Amstel 1, 1011 PN Amsterdam, The Netherlands

[3] AI Lab, Vrije Universiteit Brussel, Pleinlaan 9, 1050 Brussel, Belgium

[4] Faculty of Architecture and the Built Environment, Delft University of Technology, Jaffalaan 9a, 2600 AA Delft, The Netherlands

all infrastructural components are maintained at or above the minimum required service level and whereby unnecessary maintenance is avoided. Such strategic planning is crucial not only for the sustained economic performance of a nation, but also for safeguarding the well-being and quality of life of its populace.

The typical maintenance strategy for large infrastructural assets is to define either a proactive or reactive maintenance policy. A reactive maintenance policy entails that maintenance is only executed if the asset is almost failing or has failed. The main benefit of this strategy is that maintenance is not done unnecessarily and is therefore cost-efficient; however, Sawnson [40] describes this strategy as a fire-fighting strategy for maintenance planning because maintenance is only done if an asset is almost failing or already failed. These failures could be catastrophic with infrastructural assets, such as the collapse of the Grimburgwal quay wall in Amsterdam [24] or the deadly collapse of the Morandi bridge in Genoa [29].

A proactive maintenance policy would prevent these failures by performing maintenance so that the asset will never deteriorate to a level it can fail. This comes with the downside that proactive policies are less cost-efficient than reactive ones. How much less depends on how well the proactive policy is formulated. An example of a proactive policy is a time-based maintenance strategy, where maintenance is performed at a set interval. Time-based maintenance is easy to implement but is, in most cases, not optimal because maintenance is even performed when the asset is still in a healthy state and is, therefore, more costly [20]. Condition-based maintenance alleviates this by only doing maintenance if an asset is in a specific condition. However, this warrants sufficient asset monitoring through inspections or other methods. To improve this monitoring, predictive maintenance is applied [45], whereby the future state of an asset is predicted or when a failure will occur, minimizing the required maintenance and inspection needed, both costly operations and with scarce capacity. A prescriptive maintenance policy uses these predictions to construct an optimal maintenance plan by not only planning maintenance to prevent a single failure, but also for a longer time span in which multiple failures can occur.

Many different methods can be used for prescriptive maintenance; some examples are integer linear programming [9, 10] or genetic algorithms [8, 11, 15]. In recent years, reinforcement learning has been shown to be a promising research direction for prescriptive maintenance [4, 5, 16, 21, 25, 38]. With reinforcement learning (RL), the problem of prescriptive maintenance is formulated under a sequential decision-making setting. This means that every decision made does not just impact the immediate future but also has long-term effects. For instance, neglecting maintenance could lead to catastrophic failure

or require extensive maintenance in the future. RL learns to optimally make decisions through trial-and-error, whereby at each decision step, it receives a singular reward signal, which tells how well it is performing. The goal is to maximize the cumulative reward signal. Other reasons RL is promising for infrastructural maintenance planning is its ability to scale to larger assets, plan for decades in the future, and plan under uncertainty [4, 5].

This ability to plan under uncertainty is essential because a key insight behind prescriptive maintenance is to explicitly take uncertainty about the condition state of the asset into account while deciding the best course of action. This uncertainty stems from the fact that the structural condition of most infrastructural assets is not fully observable due to their physical locations (e.g., bridge piles being submerged in water) and/or incomplete inspection due to the complexity of the asset's components. Taking this uncertainty into account is essential in forming an effective maintenance strategy.

In addition to considering the uncertainty, it is essential to take the maintenance goal into account. This goal is, in most cases, formulated into a set of objectives for which the maintenance plan should be optimized, such as monetary cost, safety, or availability. However, comparing these objectives is a challenging task. For example, we can clearly calculate the maintenance cost for either a road or sewer pipes; conversely, we cannot quickly determine the economic benefits of performing the maintenance on those roads and sewer pipes, either due to the number of economic actors that utilize a road or that it has not a clear economic benefit in case of the sewer pipes. Furthermore, these maintenance activities aim to achieve several objectives, including ensuring serviceability for roads and sewer pipes and maintaining road availability. Therefore, optimizing infrastructural maintenance requires a multi-objective approach where these objectives are weighted against each other.

Existing RL methods for prescriptive maintenance can only optimize for a single objective, whereas most real-world maintenance optimization problems are multi-objective. Therefore, we introduce *Multi-Objective Deep Centralized Multi-Agent Actor-Critic (MO-DCMAC)* for multi-objective prescriptive maintenance planning. MO-DCMAC is based on the DCMAC [4] and MOCAC [33] algorithms for maintenance planning and multi-objective reinforcement learning, respectively. MO-DCMAC learns how to construct a maintenance plan in the same sequence as is currently done by asset managers, whereby the score is how well the maintenance plan performs according to the utility used.

If we consider infrastructural maintenance optimization as a multi-objective problem, we also need to consider what kind of scenario our method will be used. Hayes et al.

[18] describe multiple scenarios when a multi-objective approach is required. In most of these scenarios described by Hayes et al., the weighting of the objectives is not determined; however, asset managers who currently plan the maintenance know how to weigh these objectives with existing methodologies. One of these methodologies is the *Failure Mode, Effects, and Criticality Analysis (FMECA)* [7], which evaluates the utility (criticality score) of various asset failure modes based on a combination of costs, risks, and other factors such as environmental effects, all integrated in a nonlinear manner. Therefore, according to Hayes et al. [18], we should consider the *known utility function scenario* because we know how to scalarize all the objectives into a singular value.

One can assume that if we know how to weigh objectives against each other, we could use existing reinforcement learning methods for the maintenance optimization of larger assets [4, 5, 16] because we can, with the utility function, scalarize the objectives to a singular reward value. However, these existing methods are undesirable for multi-objective maintenance optimization, even with a known utility function. One reason these methods are undesirable is that they require scalarization at every training step, whereas the utility function weighs the objectives for the whole maintenance plan. Moreover, in this paper, we consider FMECA and other nonlinear methodologies as our utility function, which could lead to our maintenance optimization problem being intractable [34] if we use standard reinforcement learning for our problem.

Moreover, we formulate maintenance optimization for infrastructural assets as a *multi-objective partially observable Markov decision process model (MOPOMDP)* [39]. This MOPOMDP is formulated not only to handle uncertainty about the state of the components, but also to model different reward signals that can be taken into account with a nonlinear utility function. Additionally, we introduce a novel method for incorporating a probability value as a reward or objective. This reward value will ensure that episodic return is always a valid probability, which can be discounted. Lastly, we test MO-DCMAC on multiple environments where maintenance is planned for an infrastructural asset. For these environments, we use the historical quay walls of Amsterdam as a real-world use case (Sects. 6.1.2 and 6.1.3).

2 Background

In the background section, we discuss the key topics to understand MO-DCMAC. We begin with explaining standard reinforcement learning (RL) and deep reinforcement learning (DRL). In this part, we will also focus on actor-

critic methods, seeing that MO-DCMAC is one of them. Thereafter, we will explain multi-objective reinforcement learning (MORL) and how it differs from standard RL. For example, we will state the essential background on why standard RL methods cannot be used for our multi-objective maintenance optimization problem even though we have a known utility function.

2.1 Reinforcement learning

In the realm of maintenance optimization, addressing the challenge of determining the most effective sequence of maintenance actions over time is essential. This problem is inherently a sequential decision-making dilemma, where the objective is to optimize the scheduling and execution of maintenance activities to maximize system reliability and minimize costs. To achieve this, we want an agent to learn an optimal maintenance policy. An agent learns this by interacting within an environment, whereby the environment describes the possible maintenance actions and their effect. To frame this issue in a structured manner, we can formulate the maintenance optimization problem as a Markov decision process (MDP).

A MDP is tuple $\mathcal{M}_{\text{MDP}} = \langle S, A, T, \mathbf{R}, \gamma, H \rangle$, in which:

- $s_t \in S$ is set of states.
- $a_t \in A$ are all the possible actions.
- $T : S \times A \times S \to [0, 1]$ is the transition function that describes the probability of transitioning from state s_t to state s_{t+1} if action a_t is taken at timestep t.
- $\mathbf{R} : S \times A \times S \to \mathbb{R}$ is the reward function. The reward function maps the state s_t and the taken action a_t to a scalar reward value r_t. This reward value r_t tells how well the agent is performing.
- The discount factor $\gamma \in [0, 1]$ determines the significance of future rewards in the learning process.
- H is the horizon, which indicates how long an episode will be.

With an MDP, we want to have the agent learn policy π that maximizes the discounted cumulative reward called return:

$$\begin{aligned} \pi^* &= \arg\max_{\pi} \mathbb{E}[R_t \mid \pi, s_t = s] \\ &= \arg\max_{\pi} \mathbb{E}\left[\sum_{k=t}^{H} \gamma^k r_k \mid \pi, s_t = s\right], \end{aligned} \quad (1)$$

with π^* being the optimal policy. The expected return of policy from taking action a_t at state s_t is described through the state-action value function (Q-function):

$$Q^\pi(s, a) = \mathbb{E}[R_t \mid \pi, s_t = s, a_t = a], \quad (2)$$

with the optimal Q-function being $Q^*(s, a) = \max_{\pi} Q^\pi(s, a)$.

Similarly, the expected return from a given state s_t is formulated through the value function (V-function),

$$V^\pi(s) = \mathbb{E}[R_t \mid \pi, s_t = s]. \tag{3}$$

The aforementioned equations can be solved using tabular approaches where state-action or state value is stored for each possible state and action. However, as the number of states or actions grows, a tabular approach, like Q-learning [41], does not scale well. Deep reinforcement learning (DRL) provides a scalable solution. Deep Q-learning (DQN) [30] uses a neural network to learn the state-action value (Eq. 2). DQN and similar methods that learn a state or state-action value are called value-based methods. Another approach is policy-based methods. These methods directly learn a policy $\pi(a_t \mid s_t)$, which defines a probability distribution over the possible actions (Eq. 1). A major downside of policy-based methods is that they encounter a lot of variance during training and are thus unstable without any modifications.

One approach to reduce variance in policy-based methods is using actor-critic methods. An actor-critic method consists of two parts: the actor, which learns a policy π, and the critic, which learns to estimate the future expected returns through either a state-action (Eq. 2) or only a state (Eq. 3). Besides reducing the variance, it also enables the actor to be updated at every timestep and reduces the variance in training. Advantage actor-critic, introduced by Mnih et al. [31], further reduces the variance by updating through the advantage function, $A(s,a) = Q(s,a) - V(s)$, which measures how much better or worse an action a_t is compared to the other actions at state s_t. The advantage function can also be approximated by using only the state value function, by $A(s_t, a_t) \approx r_t + \gamma V(s_{t+1}) - V(s_t)$, allowing the critic to only use the state value function V, instead of also the state-action value function. The policy is then updated through the following loss function:

$$
\begin{aligned}
L(\pi) &= - \sum_{t=0}^{H} A(s_t, a_t) \log\left(\pi_\theta(a_t \mid s_t)\right) \\
&= - \sum_{t=0}^{H} \left(r_t + \gamma V_\psi(s_{t+1}) - V_\psi(s_t)\right) \\
&\quad \cdot \log\left(\pi_\theta(a_t \mid s_t)\right),
\end{aligned}
\tag{4}
$$

whereby θ are the weights for π, the actor, and ψ the weights for V, the critic.

2.2 Multi-objective reinforcement learning

Standard RL methods can only optimize a policy for single-objective problems, like optimizing for cost. As such, these methods fall short of many real-world problems in which multiple objectives are in play, such as infrastructural maintenance optimization. For example, in some instances, we cannot convert an objective to cost, such as safety, or it is a complex task to capture the output of asset reliability assessment methodologies, such as FMECA, to a reward function that can combine these objectives. Therefore, we will utilize *multi-objective reinforcement learning* to optimize a policy for infrastructural maintenance optimization, such that we can use these assessment methodologies in the learning process.

In multi-objective reinforcement learning (MORL), the environment and interactions are formulated as a multi-objective Markov decision process (MOMDP) [18]. This is the tuple $\mathcal{M}_{\text{MOMDP}} = \left\langle S, A, T, \vec{\mathbf{R}}, \gamma, H \right\rangle$. The only noticeable difference between a standard MDP and a MOMDP is the reward function $\vec{\mathbf{R}} : S \times A \times S \to \mathbb{R}^d$, which returns a vector of d objectives, instead of a singular scalar value. A MOMDP can be reduced to a standard MDP if $d = 1$ because the reward function would only return a singular reward value.

In standard RL, we can easily determine if a policy π is performing better than another policy π' if

$$\mathbb{E}[R_t \mid \pi, s_t = s] > \mathbb{E}[R_t \mid \pi', s_t = s]. \tag{5}$$

In contrast, this is not trivial in MORL. If we have $d = 2$ and the expected return for π is $(0, 10)$ and for π' it is $(10, 0)$, we cannot decide which policy is performing better or even if they are performing equally because one objective might be significantly more important than the other objective. This complexity in comparing objectives is evident in infrastructural maintenance optimization. For instance, is it worth improving the probability of collapse from 2% to 1% if the cost doubles? Asset managers use methodologies like FMECA to assess the best maintenance policy. Therefore, we will focus on the *utility-based approach* as described by Hayes et al. [18], so we can include methodologies such as FMECA in the training procedure.

A utility function $u : \mathbb{R}^d \to \mathbb{R}$ is a mapping of the episodic return $\vec{R} = \sum_{k=0}^{H} \gamma^k \vec{r}_k$ to a singular value. The utility function needs to be *strictly monotonically increasing*, meaning that if one of the objectives increases, the utility return can never decrease. It is nearly impossible to find an optimal policy if the utility is not monotonically increasing. This is because, in utility-based MORL, the agent aims to learn a policy that maximizes the utility. However, if the utility function is not monotonically increasing, the agent will reach a certain point during training where it can only improve the utility further if it first decreases. This will cause the agent to get stuck at a local optimum.

There are two methods of optimizing over a known utility function in MORL [18], namely the *Scalarized Expected Return* (SER) criterion, which is when the utility function is applied on the expected return:

$$\pi^* = \arg \max_{\pi} u\left(\mathbb{E}\left[\sum_{k=t}^{H} \gamma^k \vec{r}_k \mid \pi, s_t = s \right] \right). \tag{6}$$

The other criterion is the *Expected Scalarized Return* (ESR), which tries to maximize the expected return of the utility function:

$$\pi^* = \arg \max_{\pi} \mathbb{E}\left[u\left(\sum_{k=t}^{H} \gamma^k \vec{r}_k \right) \mid \pi, s_t = s \right]. \tag{7}$$

The difference is that while SER is concerned with the utility of the average outcome, ESR takes the utility over every single roll-out of the policy (and only then takes the average). When we examine most MORL methods, we see that the SER criterion is more used, while in multi-objective games, the ESR criterion is used [18]. One explanation for the ESR criterion being understudied is that it invalidates the Bellman equation if the utility is nonlinear:

$$\max_{\pi} \mathbb{E}\left[u\left(\vec{R}_t^- + \sum_{k=t}^{\infty} \gamma^k \vec{r}_k \right) \mid \pi, s_t \right] \neq$$
$$u(\vec{R}_t^-) + \max_{\pi} \mathbb{E}\left[u\left(\sum_{k=t}^{\infty} \gamma^k \vec{r}_k \mid \pi, s_t \right) \right], \tag{8}$$

where $\vec{R}_t^- = \sum_{k=0}^{t-1} \gamma^k \vec{r}_k$ is the accrued reward, until timestep t. Equation 8 shows how the Bellman equation becomes invalidated because any nonlinear utility function under the ESR criterion does not distribute across the accrued reward and the future returns [18]. Therefore, the most evident criterion to use would be the SER criterion; however, a policy learned with the SER criterion could differ significantly from one learned with the ESR criterion, as shown by Rădulescu et al. [36]. Moreover, Hayes et al. [18] describe when ESR or SER should be used. SER is more flexible because it takes the expected return of the objectives as input for the utility function. This means that in our context of maintenance optimization, an asset might collapse in some runs so long it is safe on average. ESR is stricter because it optimizes the expected return of the utility function, which means that at every run, the asset should not collapse because, otherwise, the utility will be low. This strictness of the ESR criterion is more desired for maintenance optimization because an asset should never collapse.

3 Related work

In infrastructural maintenance optimization, we can optimize either at the network level [15, 21] or the asset level [3–5, 16]. At the network level, we optimize the maintenance for multiple assets, whereby each maintenance action is applied to the whole structure. When we plan maintenance at this level, we mainly consider the interaction between the assets. One of these interactions could be to ensure that destinations are reachable in a road network [15] or by rewarding the grouping of maintenance actions of assets close together [21]. At the asset level, the level of our problem, we only consider a single asset but plan the maintenance of its components. Similar to the network level, we consider interactions, but instead between those components, such as a higher risk of failure when multiple critical components are failing.

When we compare other research, we notice that most asset level maintenance optimization problems are formulated as a POMDP. An example of this is DCMAC, by Andriotis and Papakonstantinou [4], which uses a discrete state space for the health level of each component, whereby the belief is calculated through a Bayesian update. It is also possible to use a continuous state space with a POMDP; however, this introduces increased complexity because the belief cannot be updated anymore through a Bayesian update. The reason for this is that it requires an integral that has no closed-from solution. Nevertheless, it is still possible to use a continuous POMDP [16, 25, 38], in combination with approximation methods for the belief, such as Kalman filters [22], particle filters [13], other deep learning methods [17], or other methods. However, in this paper, we will only consider a discrete state space and thus not require any of these methods.

Many methods are used for maintenance optimization besides deep reinforcement learning. Common methods are integer linear programming [9, 10], genetic algorithms [8, 11, 15] and other optimization methods [43]. Nevertheless, these methods do not consider partial observability and struggle with scalability.

This struggle with scalability is also the reason why we focus on DRL methods instead of other methods, such as non-deep RL, that can handle partial observability, such as POMCP [37] or even methods for MOPOMDP [35]. In comparing the benchmarks used by both methods, we found that Roijers et al. [35] used benchmarks with significantly smaller state and action spaces. Some of the benchmarks used by Silver and Veness [37] had a larger state space than we used in our experiments; however, the action space is magnitudes smaller, with at most 100 possible actions, while our smallest environment has 13122 possible action combinations and the largest approximately

1.88×10^{11}. Therefore, the Monte Carlo tree search used by Silver and Veness is infeasible with such a large action space.

This issue with the action space, or the "curse of dimensionality" of the action space, is a common problem when it comes to using RL for maintenance optimization problems because the agent should not only take a singular action for each timestep, but also rather an action for each component, such that the action space grows exponential to the number of components. In our research, we notice that two kinds of methods are used to address this issue. The most common method is to model it as a multi-agent problem, such as DCMAC [4]. In it, Andriotis and Papakonstantinou use a factorized action space representation and model the neural network such that it has a shared input and a separate output head for each agent/component. The shared input is not always necessary, and DDMAC uses a separate input for each component [5, 32]. However, this has the limitation that there is no communication between the agents/components, and each action will be taken independently of the states of other components. Another more recent approach is to use hierarchical reinforcement learning (HRL) [3, 16] for tackling larger action spaces by decomposing them into smaller action spaces using HRL. However, we focus more on the factorized action space approaches in this paper.

In most MORL research, we see the focus on methods for training a Pareto Frontier of different policies [1, 44]. However, these objectives are weighted linearly, through different weights, and use the SER criterion. Nevertheless, these methods are useful if the utility is unknown. These Pareto Frontier methods are already used for use-cases, such as the job-shop scheduling problem [28] and manufacturing [12].

As described by Hayes et al. [18], the ESR criterion is understudied, which is mainly due to the increased complexity that comes with the ESR criterion and a known nonlinear utility function. One method for this is multi-objective categorical actor-critic (MOCAC) [33]. During our research on applying ESR criterion deep MORL methods like MOCAC, we could not find any existing papers. Hence, to the best of our knowledge, this paper is the first time a deep MORL method with the ESR criterion has been employed to solve a real-world problem, such as optimizing infrastructural maintenance.

4 Maintenance optimization problem

Before we explain our proposed method, MO-DCMAC, we first state the problem formulation of the maintenance optimization problem, for which we want to find an optimal policy and how this is formulated as a *Multi-Objective Partially Observable Markov decision process* (MOPOMDP) [39]. In our explanation, we will also include how a MOPOMDP differs from a MOMDP and MDP.

4.1 Problem formulation

An asset consists of N components, whereby each component $c_i, i \in \{0, 1, \ldots, N - 1\}$ can belong to one of $K \in \mathbb{N}$ component groups. The health state of each component c_i is represented by an integer variable $s_t^i \in \{0, 1, \ldots, |S| - 1\}$, where $|S|$ is the number of possible health states of the component. Each component c_i has a current degradation rate τ_t^i that can be increased until $\tau_{\max}$. The degradation rate τ_t^i dictates the probability that component c_i transitions from a healthy to a non-healthy state. Besides the degradation rate, to what component group $c_i \in K$ belongs also influences the transition probability. This means some component groups might degrade faster than others, even if their degradation rate τ is higher.

For each component c_i, we can take an individual maintenance action $a_t^{c_i}$, which is the action taken at timestep t. Each maintenance action $a_t^{c_i}$ will always increase the health state, except if the action is do nothing. If replaced, the degradation rate of the component c_i is set back to 0. Besides taking an action for each component, we can also take an action $a_t^{\#}$ for the whole asset. These *global* actions will affect all the asset components and could be, for example, an inspection. Moreover, each action, for the component and global, will have a cost c, which is the monetary value of that action.

This maintenance optimization problem is *partial observable*, so rather than observing the state $s_t^{c_i}$ of component c_i, we instead observe an observation $o_t^{c_i} \in \mathcal{O}$, where $\mathcal{O}$ is the set of possible observations. The observation $o_t^{c_i}$ is dependent on both the current health state $s_t^{c_i}$ of component c_i as well as action taken at time t. For example, if a global inspection is done at timestep t, the next observation will fully reveal the state, meaning that $o_{t+1} = s_{t+1}$; however, if no inspection is done, but the component c_i is repaired, then only the state is revealed for component c_i. If neither an inspection nor a maintenance action is done, the observation will be a low-accuracy representation of the current state. This accuracy depends on the specific maintenance environment.

Within the K components groups, multiple sub-dependency groups can exist $\mathcal{G}_i^k$. Within the sub-dependency groups, there may be one or more components from the same component group. When one or more $c \in \mathcal{G}_i^k$ are in the worst state $|S| - 1$, a probability $F(\mathcal{G}_i^k)$ is returned that signifies the effect that $\mathcal{G}_i^k$ has on *probability of collapse*,

which increases as more components are failing in the same component group. For example, if components $c_i \in \mathcal{G}_i^k$ and $c_j \in \mathcal{G}_i^k$ are both failing, then $F(\mathcal{G}_i^k)$ will be higher. However, if $c_j \notin \mathcal{G}_i^k$, then $F(\mathcal{G}_i^k)$ would not increase because component c_j does not belong to the same component group.

To calculate the probability of collapse ρ_t for a single time step, we use the following equation that combines the effect of each component group on the probability of collapse:

$$\rho_t = 1 - \prod_{k \in K} \prod_{i=0}^{|\mathcal{G}^k|-1} \left(1 - F(\mathcal{G}_i^k)\right). \tag{9}$$

The objective is to find an optimal maintenance strategy, i.e., the series of maintenance actions until the horizon H that minimizes both the total *cost* $\mathcal{C}$ and the *probability of collapse* ρ of the asset.

4.2 Multi-objective partially observable Markov decision process (MOPOMDP)

In our problem formulation, we stated that the real health state of each component is only *partially observable*. Any agent would likely not learn an optimal policy if we would formulate the problem with an MOMDP. Therefore, we must formulate the problem as *Multi-Objective Partially Observable Markov decision process* (MOPOMDP) [39]. A MOPOMDP is a tuple $\mathcal{M}_{\text{MOPOMDP}} = \langle S, A, \Omega, T, O, \vec{\mathbf{R}}, \gamma, H \rangle$, whereby a MOPOMDP adds the following:

- $o_t \in \Omega$ is the set of observations that the agent can receive.
- $O : S \times A \times \Omega \rightarrow [0, 1]$ is the observation function and describes the probability that an observation is received based on the current state and action.

In a MOPOMDP, the agent receives an observation o_t at each timestep t instead of the current state s_t, whereby the observation is an inaccurate representation of that state. For example, sensors are inaccurate and could give different readings for the same state. Within a MOPOMDP, the sensor would be the observation function, and the sensor outputs the observation the agent would receive.

Due to the inaccuracy of the observation, a standard RL method will most likely result in a sub-optimal policy. A solution for this is to add some form of memory that keeps a belief over the current state $\mathbf{b}_t$. If the state space is discrete, the belief can be formulated as a vector of size $|S|$,

whereby the belief is the probability that the agent is in that given state. This belief is then updated at each timestep through a Bayesian update:

$$
\begin{aligned}
b(s_{t+1}) &= P(s_{t+1} \mid o_{t+1}, a_t, \mathbf{b}_t) \\
&= \frac{P(o_{t+1} \mid s_{t+1}, a_t)}{P(o_{t+1} \mid \mathbf{b}_t, a_t)} \\
&\quad \sum_{s_t \in S} P(s_{t+1} \mid s_t, a_t) b(s_t)
\end{aligned}
\tag{10}
$$

In the belief update, shown in Eq. 10, the belief $\mathbf{b}_t$ is updated through multiple mechanics. One of them is through the known transition function $P(s_{t+1} \mid s_t, a_t)$, whereby even though if do not get any more information, add some more accurate information about the state. Secondly, the current action a_t can also influence what kind of observation is received at the next timestep $t + 1$, such as doing an inspection, which could give a more accurate observation.

The specific definition of our MOPOMDP is as follows:

- $s_t \in S$ The state space is discrete, with the N components being able to be in one of $|S^{c_i}| = 5$ states and a degradation rate $\tau_t^{c_i}$ for each component, whereby the maximum degradation rate is $\tau_{\max} = 50$
- $a_t \in A$ The action space is the combination of actions for the N component, which is $A^{c_i} \in \{donothing, repair, replace\}$ with $1 \le i \le N$, and the global action for the whole structure, which is $A^{\#} \in \{donothing, inspect\}$.
- $o_t \in \Omega$ The observation space is equal to the state space, except τ^{c_i}; in that the observation for each component can be one of $|S^c|$ states.
- $T : S \times A \times S \rightarrow [0, 1]$ At each timestep t, the component has a probability of either staying in a specific state or transitioning to a worse state, based on $\tau_t^{c_i}$ and the associated transitioning matrix. If a *repair* action is taken, the component condition is improved, which is depicted by moving one state up, and if the *replace* action is taken, the component is set to the best state, and $\tau_{t+1}^{c_i}$ is set to 0.
- $O : S \times A \times \Omega \rightarrow [0, 1]$ The received observation is based on $a_t^{\#}$. If the *nothing* action is taken, the observation will be of low-accuracy representation of the state. This low-accuracy observation only indicates if a component is either in the first two states or the last three states. Still, it cannot differentiate between those specific states. If the *inspect* action is taken, the state will be fully observed. Moreover, the state of a specific component c_i is also fully observed if c_i if a *repair* or *replace* action is taken for c_i.

- $\vec{R} : S \times A \times S \to \mathbb{R}^2$ The reward vector consists of two values $\vec{r}_t = \{-\mathcal{C}_t, \log(1 - \rho_t)\}$. $-\mathcal{C}_t$ represents the *cost*, which has been made negative because a lower cost is preferable. $\log(1 - \rho_t)$ is the reward value for the *probability of collapse*. This reward value represents the inverse log probability of collapse occurring at timestep t. Using this log probability as our reward value, we can calculate the probability of collapse of the whole episode as input for our utility function. We explain this in-depth later in this section.

- H The horizon of any maintenance optimization is the time window and interval when maintenance should be planned. For example, if maintenance is planned yearly for 50 years, $H = 50$. If the interval is monthly, then the horizon for 50 years is $H = 600$. However, in this paper, $H = 50$ is used because we only consider yearly intervals for a time window of 50 years.

Due to the partial observability of the state, we introduce a belief $\mathbf{b}_t$ over the states. We update this belief with the newly received observation o_t through a Bayesian update as seen in Eq. 10 [4].

In our MOPOMDP formulation, we stated that the *probability of collapse* reward $\log(1 - \rho_t)$ as the inverse log probability of collapse occurring at timestep t. Using this inverse log probability is based on two key considerations. Firstly, using the log probability ensures that the episodic return of the probability of collapse $R_\rho = \sum_{t=0}^{H} \log(1 - \rho_t)$ is a valid probability value. Secondly, R_ρ will be the log probability of no collapse occurring during the episode; therefore, we can use the base of the natural logarithm e to obtain the probability of collapse over the whole episode easily:

$$
\begin{aligned}
P_\rho &= 1 - \prod_{t=0}^{H}(1 - \rho_t) \\
&= 1 - e^{\sum_{t=0}^{H} \log(1 - \rho_t)} \\
&= 1 - e^{R_\rho}.
\end{aligned}
\tag{11}
$$

5 Multi-Objective Deep Centralized Multi-Agent Actor-Critic (MO-DCMAC)

In the previous section, Sect. 4, we discussed the formulation of the maintenance optimization problem, for which we want to learn a maintenance strategy and how we formalized this as a MOPOMDP. However, standard RL methods face significant challenges in learning an optimal policy, primarily due to two essential reasons.

The first is the curse of dimensionality in the action space. The action space grows exponentially to the number of components and possible maintenance actions; for example, with 12 components and three possible actions, the action space would be $3^{12} = 531441$. Andriotis and Papakonstantinou introduced *Deep Centralized Multi-Agent Actor-Critic* (DCMAC) [4], an actor-critic method, to alleviate this. DCMAC uses a factorized action space representation, whereby the actor network has an output for each component that outputs an action for that component.

Secondly, our maintenance optimization problem is a multi-objective problem with a known nonlinear utility function. Moreover, we want to learn a policy using the ESR criterion (Eq. 7); however, this introduces significant complexity because if the utility function is nonlinear, then the Bellman equation becomes invalidated (Eq. 8) [18]. Reymond et al. introduced *Multi-Objective Categorical Actor-Critic* (MOCAC) [33] that utilizes a distributional critic, which learns a multi-variate distribution over the returns of the objective. MO-DCMAC requires this multi-variate distribution over the return, introduced by Reymond et al. [33], to be able to learn an optimal policy with the ESR criterion in a multi-objective problem setting. We explain this further in this section.

Our proposed method, *Multi-Objective Deep Centralized Multi-Agent Actor-Critic (MO-DCMAC)*, combines DCMAC [4] and MOCAC [33]. Combining these methods allows us to overcome the curse of dimensionality of action space by using the actor network of DCMAC and the ability to directly optimize over a nonlinear utility function with the ESR criterion by using the critic network of MOCAC.

The goal of MO-DCMAC is to learn a policy that returns the highest utility for a given multi-objective maintenance optimization problem with a known nonlinear utility function under the ESR criterion. We needed to integrate DCMAC [4] and MOCAC [33] into MO-DCMAC to achieve this goal. The critic of MO-DCMAC learns the multi-variate distribution of the objectives returns instead of the expected return because to learn under the ESR criterion; we require the return as input for the utility. Moreover, this allows MO-DCMAC to correctly calculate the advantage A_t, which is used to update the actor. The interaction between the critic and actor can be seen in Fig. 1 and Algorithm 1. The exact implementation details will be explained in the remainder of this section.

Algorithm 1 MO-DCMAC

Require: η Update Steps, $n_{\max}$ Number of Training Steps
1: Initialize actor weights θ and critic weights ψ.
2: Initialize step counter $n = 0$.
3: **while** True **do**
4:     Get s_0
5:     $\mathbf{b}_0 \leftarrow b(s_0)$
6:     **for** $t \leftarrow 1$ **to** H **do**
7:         Select actions $a_t = \left\{ a_t^{c_0}, \ldots, a_t^{c_{N-1}}, a_t^{\#} \right\} \sim \pi_\theta(\cdot \mid \mathbf{b}_t, \frac{t}{H}, \vec{R}_t^-)$.
8:         Perform a_t and receive $\vec{r}_t$ and observation o_{t+1}.
9:         Calculate the new belief $\mathbf{b}_{t+1} \leftarrow b(s_{t+1})$.
10:         Store the tuple $((\mathbf{b}_t, \frac{t}{H}, \vec{R}_t^-), a_t, \vec{r}_t, (\mathbf{b}_{t+1}, \frac{t+1}{H}, \vec{R}_{t+1}^-))$
11:         $n \leftarrow n + 1$
12:         **if** $n \mod \eta = 0$ **then**
13:             Get previous η training examples.
14:             Calculate advantage using Equation 15.
15:             Calculate critic loss using Equations 17 and 18.
16:             Calculate policy loss with Equation 20.
17:             Calculate final loss with Equation 21.
18:             Reset the buffer.
19:         **end if**
20:         **if** $n \geq n_{\max}$ **then**
21:             **return**
22:         **end if**
23:     **end for**
24: **end while**

The critic and most of the loss calculations in MO-DCMAC are based on previous work introduced by Reymond et al. [33]. Reymond et al. introduce a distributional critic that learns the multi-variate distribution Z of the return of the objectives. They based this on the work of Bellemare et al. [6], whereby the major difference is that Reymond et al. [33] learn a multi-variate distribution. The reason why the critic learns the distribution of the objectives instead of the expected vectorial return is that when the expected vectorial return is scalarized by a utility function u, it would learn under the SER criterion because the vectorial value function learns the expected return (Eq. 3). To ensure that MO-DCMAC can learn an optimal

policy under the ESR criterion, episodic return is required instead [33].

Therefore, we utilize the critic introduced by Reymond et al. [33] to learn a distribution Z through the following equation:

$$V(s_t) = \sum_{i \in [\{0,\ldots,\mathcal{N}-1\}]^d} \vec{z}_i Z_\psi(\vec{z}_i \mid s_t). \tag{12}$$

where $\vec{z}_i$ is the support atom for the vectorial return. Each atom $\vec{z}_i$ for $\forall i \in [\{0,\ldots,\mathcal{N}-1\}]^d$ is calculated by,

$$\vec{z}_i = \vec{V}_{\min} + i\Delta\vec{z}$$
$$\text{with } \Delta\vec{z} := \frac{\vec{V}_{\max} - \vec{V}_{\min}}{\mathcal{N} - 1} \tag{13}$$

This support atom is calculated by providing $V_{\min}, V_{\max} \in \mathbb{R}$ for each objective, which represents the minimum and maximum return for that objective. Therefore, $\vec{z}$ represents the possible returns we can have for the objectives. The critic $Z_\psi(\vec{z}_i \mid s_t)$ learns the probability that any given return in $\vec{z}$ will occur from the current state s_t. $\vec{z}$ itself consists of $\mathcal{N}^d$ atoms, where each atom represents a possible return of the objectives. The number of atoms $\mathcal{N}$ can be chosen arbitrarily; however, Reymond et al. [33] assume the same number of atoms for each objective.

Due to Eqs. 12 and 13 introduced by Reymond et al. [33], the critic learns the discretized distribution of the objectives instead of the expected return. Because $\vec{z}$ does not represent the expected return but is defined as a return, MO-DCMAC can learn under the ESR criterion. Therefore, in combination with the accrued reward $\vec{R}_t^- = \sum_{k=0}^{t-1} \gamma^k \vec{r}_k$, we can calculate a preference score u_t [33] under the ESR criterion by:

$$u_t = \sum_{i \in [\{0,\ldots,\mathcal{N}-1\}]^d} u\big(\vec{R}_t^- + \vec{z}_i\big) Z_\psi(\vec{z}_i \mid s_t). \tag{14}$$

With the preference score u_t being calculated through Eq. 14, the advantage $A(s_t, a_t)$ of MO-DCMAC is calculated through Eq. 15 [33]

$$A(s_t, a_t) = \left(\sum_{i \in [\{0,\ldots,\mathcal{N}-1\}]^d} u\big(\vec{R}_t^- + \gamma^t(\vec{r}_t + \gamma\vec{z}_i)\big) \cdot Z_\psi(\vec{z}_i \mid s_{t+1}) \right) - \left(\sum_{i \in [\{0,\ldots,\mathcal{N}-1\}]^d} u\big(\vec{R}_t^- + \gamma^t\vec{z}_i\big) \cdot Z_\psi(\vec{z}_i \mid s_t) \right) \tag{15}$$

Reymond et al. [33] did not originally normalize the advantage. However, we found that normalizing

significantly increased training stability; therefore, the final advantage used is:

$$\hat{A}(a_t, s_t) = \frac{A(a_t, s_t) - \mu_A}{\sigma_A}, \tag{16}$$

where μ_A is the mean advantage of the current batch and σ_A is the standard deviation of the advantage of the current batch.

For the critic's loss calculation, first the distributional Bellman update $\hat{\mathcal{T}}\vec{z}_j = \vec{r}_t + \gamma z_j$ for each atom $\vec{z}_j$ is calculated. Reymond et al. [33] then distributes the probability of $Z_\psi(\vec{z}_i \mid s_{t+1})$ to the immediate neighbors of $\hat{\mathcal{T}}\vec{z}_j$, whereby the projected update is:

$$\Phi\hat{\mathcal{T}}Z_\psi(s_t) = \left(\sum_j \left[1 - \frac{\left| \left[\hat{\mathcal{T}}\vec{z}_j \right]_{V_{min}}^{V_{max}} - \vec{z}_i \right|}{\Delta z} \right]_0^1 \right.$$
$$\left. \cdot Z_\psi(\vec{z}_i \mid s_{t+1}) \right), \tag{17}$$
$$\forall i \in [\{0, \ldots, \mathcal{N} - 1\}]^d,$$

with $[\cdot]_b^a$ meaning that it is bounded between a and b.

The loss of the critic is then calculated with Eq. 17 through the cross-entropy term of the KL divergence:

$$\mathcal{L}^V(V) = D_{KL}\left(\Phi\hat{\mathcal{T}}Z(s_t) \parallel Z(s) \right) \tag{18}$$

Previously, we stated that we based the actor architecture on DCMAC [4]. The main contribution of DCMAC is the factorized action output, where for each component c_i, there is a distinct output layer and, depending on the setup, one or more hidden layers.

DCMAC is an actor-critic method, which calculates the loss for the actor similarly to Eq. 4. However, the main difference is that DCMAC has to combine the output of all the output heads in the actor network for the loss calculation, yielding:

$$\mathcal{L}^\pi(\pi) = -w_t \left(\sum_{j=0}^{N-1} \log\left(\pi_\theta^{c_j}\left(a_t^{c_j} \mid \mathbf{b}_t \right) \right) \right) A(\mathbf{b}_t, a_t), \tag{19}$$

where $w_t = \min\left(q, \frac{\pi(a_t \mid s_t)}{\mu(a_t \mid s_t)} \right)$ is a truncated importance sampling [42] to address the high variance that log summation may introduce, c_j the component for which an action is selected, and $\mathbf{b}_t$ is the belief at timestep t because DCMAC is used to find an optimal policy in a POMDP. The reason a truncated importance sampling is needed is that DCMAC is an off-policy method because it collects training examples in a replay buffer and samples during training from it. In contrast, MO-DCMAC is an on-policy method, meaning we do not reuse training examples similar to MOCAC [33]. Therefore, the importance sampling is not necessary anymore due to $\frac{\pi(a_t \mid s_t)}{\pi(a_t \mid s_t)} = 1$. Additionally, we introduce another separate head that selects a global action head from $A^\#$. Therefore, the modified policy loss calculation will be:
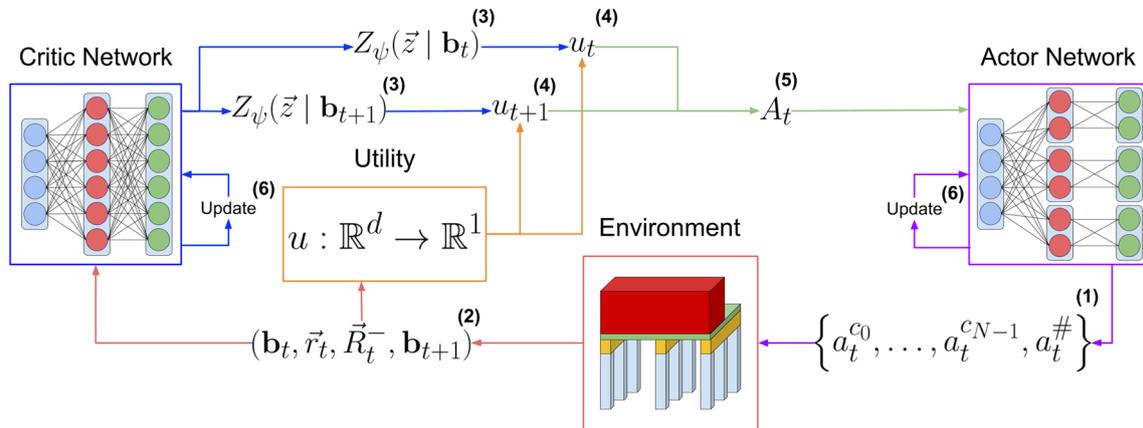

Fig. 1 Training sequence for MO-DCMAC, whereby the blue nodes are the input layers, the red hidden layers, and the green output layers. Algorithm 1 is the pseudocode of MO-DCMAC and shows how each equation is used. Step (1) involves the actor sampling actions for each component and a collective global action from the current belief state $\mathbf{b}_t$. In step (2), these actions are processed by the environment to determine the subsequent belief state $\mathbf{b}_{t+1}$ and to generate the reward vector $\vec{r}_t$. This reward is then used with the accrued reward $\vec{R}_t^-$ for further calculations. Step (3) has the critic network predicting the joint distribution of the returns across objectives, denoted as $Z_\psi(\vec{z} \mid$ $\mathbf{b}_t)$ for the current state and $Z_\psi(\vec{z} \mid \mathbf{b}_{t+1})$ for the next. In step (4), these predictions, along with the reward vector $\vec{r}_t$ and accrued reward $\vec{R}_t^-$, are used to compute the preference scores for the current u_t and next state u_{t+1}. Step (5) employs these preference scores to calculate the advantage of the state A_t. Finally, step (6) updates the critic and actor. The overall loss combines actor and critic losses with an entropy bonus. Step (6) can be done at every step but can also be done at every η step, whereby the actor and critic are updated from the previous η steps

$$\mathcal{L}^{\pi}(\pi) = -\Bigg(\sum_{j=0}^{N-1} \log \left(\pi_\theta^{c_j} \left(a_t^{c_j} \mid \mathbf{b}_t \right) \right)$$

$$+ \log \left(\pi_\theta^{\#} \left(a_t^{\#} \mid \mathbf{b}_t \right) \right) \Bigg) \hat{A}(\mathbf{b}_t, a_t), \tag{20}$$

where $\hat{A}(s_t, a_t)$ is calculated through Eqs. 15 and 16. Similar to MOCAC [33], we combine the critic loss and the policy loss through,

$$\mathcal{L}(\pi, V) = \mathcal{L}^{\pi}(\pi) + \lambda_{\text{val}} \mathcal{L}^{V}(V) - \lambda_{\text{ent}} \mathcal{H}(\pi), \tag{21}$$

where $\mathcal{H}(\pi)$ is the entropy bonus from π, λ_{val} the coefficient for the value loss and λ_{ent} the coefficient for the entropy bonus. Both λ_{value} and λ_{ent} are hyperparameters and control the contribution of the value and entropy loss, respectively.

MO-DCMAC takes three input values, namely $\mathbf{b}_t$, $\frac{i}{H}$ and $\vec{R}_t^-$. The first value is the belief $\mathbf{b}_t$ at timestep t, which we update through Eq. 10, $\frac{t}{H}$ is the normalized timestep, with H being the horizon of the episode. Lastly, we included the accrued reward $\vec{R}_t^-$ as necessitated by the multi-objective setting under the ESR with a nonlinear utility [33].

6 Experimental setup

This section explains the details of the experiments we conduct to show the capabilities of our introduced method, MO-DCMAC, to minimize the cost and probability of collapse through a given utility function that weighs these two objectives. We start by introducing the environments we used in our experiments. Afterward, we explain the utility functions used to train MO-DCMAC and state the baseline policies we used to evaluate the performance of MO-DCMAC.

6.1 Environments

We test the performance of MO-DCMAC in three different maintenance environments, namely: (i) Simple homogeneous asset, in which components are of the same type, and the calculation of the probability of collapse has no dependencies. (ii) Quay wall asset environment has multiple types of components and dependencies. This environment is based on the inner-city quay wall of Amsterdam. (iii) Larger quay wall asset, a larger version of the above environment, which has more components and dependencies.

6.1.1 Simple Homogeneous Asset

In the simple homogeneous asset environment, maintenance is planned for an asset with eight components of the same type and has the same degradation rate. The effect of a failing component is also independent of other failing components, meaning that there are eight sub-dependency groups because each sub-dependency group has one component. The full details of this environment can be found in "Appendix Simple Homogeneous Asset."

6.1.2 Quay Wall Asset

The second environment is the quay wall asset environment. This environment is based on the historical inner-city quay walls of Amsterdam.[1] [24]. Figure 2 shows the components of this environment. In this environment, we only plan the maintenance of the wooden components because these components deteriorate significantly faster than the brickwork and are more critical for structural integrity. We considered three types of components: Wooden Beams, Wooden Kesps, which connect to the Wooden Beams, and the Wooden Floor, which rests on the Wooden Kesps. In this environment, we considered dependencies. For instance, the Wooden Beams connected to the same Wooden Kesp belong to the same sub-dependency group. The contribution to the probability of collapse is almost negligible if one Wooden Beam is in the worst state; however, if two or more are, the contribution to the probability of collapse is significantly higher due to the less support this group of Wooden Beams can give to the whole quay wall. More information, such as the degradation matrices and the specific groups, are found in "Appendix Quay Wall Asset."

6.1.3 Larger Quay Wall Asset

The last environment we consider is the larger quay wall asset. This environment is almost identical to the previous one, except that this environment doubles the total components. We use this environment to test how well MO-DCMAC scales to the number of components. The exact specifications of this environment can be found in "Appendix Larger Quay Wall Asset."

6.2 Utility functions

We test each of the environments with two different utility functions: a threshold utility function, where the probability of collapse should always be below a certain threshold; otherwise, a penalty will be received, and a utility function based on the failure mode, effects, and criticality analysis (FMECA) methodology, which asset

managers use to determine the desired effect of the maintenance strategy.

As previously discussed, the input for the utility functions is the episodic return of cost $R_\mathcal{C} = \sum_{t=0}^{H} \gamma^t (-\mathcal{C}_t)$ and the log probability of collapse $R_\rho = \sum_{t=0}^{H} \gamma^t \log(1 - \rho_t)$, where the log probability of collapse R_ρ is calculated to P_ρ (Eq. 11). This probability value P_ρ is the probability that during the planned period, a collapse can occur.

In our results, we examine how these utilities lead to different policies by comparing the differences in the objective scores, cost $\mathcal{C}$, and probability of collapse ρ.

6.2.1 Threshold utility

The threshold utility function is based on standard safety requirements set for infrastructural assets, namely that an asset should adhere to a minimal safety [27]. With the threshold utility, MO-DCMAC should learn a policy that optimizes for cost while ensuring the probability of collapse will not exceed a certain threshold, such that the asset adheres to the minimal safety requirements. Minimizing the probability of collapse beyond the threshold will not improve the utility score. Therefore, MO-DCMAC should find an optimal policy that ensures the proposed maintenance plan never exceeds the threshold but does not drastically increase cost. We formulated the utility as follows:

$$u(R_\mathcal{C}, P_\rho) = \begin{cases} R_\mathcal{C} & \text{if } P_\rho \le 0.1 \\ 3(R_\mathcal{C} + 1) & \text{if } 0.1 < P_\rho \le 0.2 \\ 5(R_\mathcal{C} + 2) & \text{otherwise} \end{cases} \quad (22)$$

In our experiments, a penalty is received if $P_\rho > 0.1$, which is increased if it is $P_\rho > 0.2$. However, these thresholds can be higher or lower, or the penalty amount can be changed depending on the asset, with the only restriction that the utility should be monotonically increasing.

In our experiments, we show the effect of the discount factor γ on achieving an optimal policy with threshold utility; namely, if the discount factor is too low, it could result in a learned policy that believes it is below the threshold because the discounted return of the probability of collapse is. Still, it is not because the non-discounted return is above the threshold.

This difference between the non-discounted and discounted returns is not a problem for standard RL. However, for utility-based MORL, this difference could be because we optimize over the utility function. This is especially essential for utility functions that can contain a strict

threshold for an objective and where optimizing the objective is not beneficial for the utility score if the threshold has been met, such as the threshold utility. Therefore, we will test the following discount factors: 0.9, 0.975, 0.99, 0.995, and 1. These experiments are conducted with the *Simple Homogeneous Asset* environment (Sect. 6.1.1).

6.2.2 FMECA utility

The FMECA utility function is based on the *Failure Mode Effect and Criticality Analysis (FMECA)* methodology [7]. The goal of FMECA, in a maintenance planning setting, is to determine the most optimal maintenance strategy that limits the probability that a failure mode occurs while ensuring that other objectives are also optimized. Within FMECA, a wide range of objectives can be used. In our setting, we will focus only on our existing objectives: probability of collapse and cost; however, this utility can be extended to include other objectives.

The FMECA methodology we use in this paper is based on the one used by asset managers in Amsterdam for the inner-city quay walls. These asset managers give a score to each objective based on the effect of a proposed maintenance strategy on that objective. Each objective is classified into one of six bins to compute its utility score. The first five bins incrementally assign scores from one to five, while the last bin gives a score of ten; lower scores are preferable. Notably, each bin's range decreases logarithmically from the worst to the best bin. However, this binning approach can potentially prolong MO-DCMAC's training time. It may either trap MO-DCMAC in a local optimum, taking longer to escape, or, in the worst-case scenario, prevent escaping altogether. Therefore, we used a logarithm for smoother score growth to mitigate the risk of this happening (Fig. 3). We formulate the FMECA utility as follows:

$$c_{\text{utility}}(R_\mathcal{C}, C_{\max}) = 6 \log_{10}\left(1 + 10 \frac{R_\mathcal{C}}{C_{\max}}\right) \\ + \text{Pen}(R_\mathcal{C}, C_{\max}) \quad (23)$$

$$f_{\text{utility}}(P_\rho, F_{\max}) = 6 \log_{10}\left(1 + 10 \frac{P_\rho}{F_{\max}}\right) \\ + \text{Pen}(P_\rho, F_{\max}) \quad (24)$$

$$\text{where } \text{Pen}(x, x_{\max}) = \begin{cases} 4 & \text{if } x \ge x_{\max} \\ 0 & \text{otherwise} \end{cases} \quad (25)$$

Equations 23, 24, and 25 are used to calculate the FMECA score of each objective. We employed logarithm to base 10 ($\log_{10}$) to match the original FMECA scoring, where binning was used to determine the score and where the bins range decreased logarithmically. This logarithmic scoring
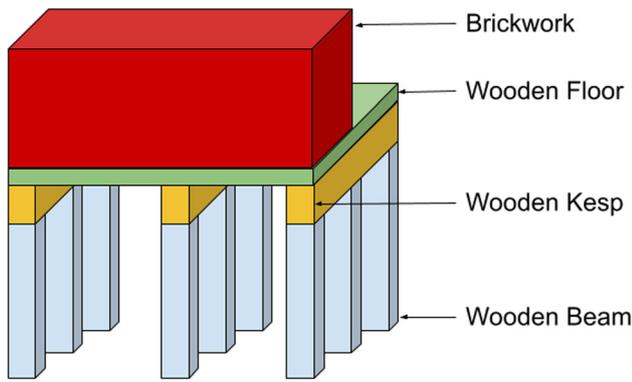
Fig. 2 Representation of the quay wall asset environment. Maintenance will only be done for the wooden components

alone is sufficient to simulate the first five bins that score from one to five. However, we had to add Eq. 25 to achieve the correct behavior related to the last bin, where a score of ten is assigned. This penalty is added if either the cost R_C is higher than C_{max} or the probability of collapse is higher than F_{max}, such that the objective score will be increased from six to ten. For our experiments, we set $F_{max} = 0.2$ and $C_{max} = 4$. However, lower or higher values are possible because F_{max} and C_{max} act as weights within the FMECA utility.

$$u = -(\max(1, c_{utility}(R_C, C_{max})) \\ \cdot \max(1, f_{utility}(P_\rho, F_{max}))) \quad (26)$$

Equation 26 calculates the last step of the utility and combines the score calculated through Eqs. 23 and 24. We make the utility negative because, within the FMECA method, a lower FMECA score is preferred. We also ensure that u never can be lower than 1, which is the minimum score in FMECA. Moreover, if we would not set both $c_{utility}$ and $f_{utility}$ to at least 1, MO-DCMAC would likely learn to do no maintenance at all because when the cost is 0, the utility score would also be 0.

6.3 Baseline policies

In our experiments, we will compare MO-DCMAC to baseline policies based on current maintenance strategies. In this subsection, we will explain each of these baseline policies.

- Year-Based Action (YBA) policy. This policy does no inspection and will take a *repair* or *replace* action for each component on year interval.
- Year-Based Inspection and Condition-Based Action (YBI-CBA) policy. The YBI-CBA policy inspects at a set interval whereby maintenance action is taken at the next time step based on the inspection.

- Condition-Based Inspection and Condition-Based Action (CBI-CBA) policy. The CBI-CBA policy is similar to the YBI-CBA policy; however, the inspection is not done at an interval but when a percentage of the components are in a worse state. A component is in a worse state if it is in one of the three worst states. Without an inspection, we can only detect if a component is in the first two or the last three states, but we cannot determine the exact state.

The CBA policies are set to *do nothing* if a component is in the first state, *repair* if in the second or third state, and *replace* if in the fourth or last state. We optimized the CBI, YBI, and YBA policies by either finding the optimal yearly interval at which an inspection should be done for YBI or repairing or replacing should be done for YBA. We optimized CBI for the percentage of components that should be in a worse state before an inspection is done. Our results will only show the performing policy for each environment and utility.

We did not include other single-objective RL methods or multi-objective RL methods since they cannot learn optimal policies for the given problem setting. Reymond et al. [33] showed that single-objective RL methods trained on only the utility score, given at the last state, result in highly unstable policies that always underperform. Other multi-objective RL methods cannot learn with the ESR criterion [18], and therefore, we cannot compare the results of MO-DCMAC with the results of these methods. We did not include MOCAC [33] since it would require substantial adjustments for it to be able to function in our problem
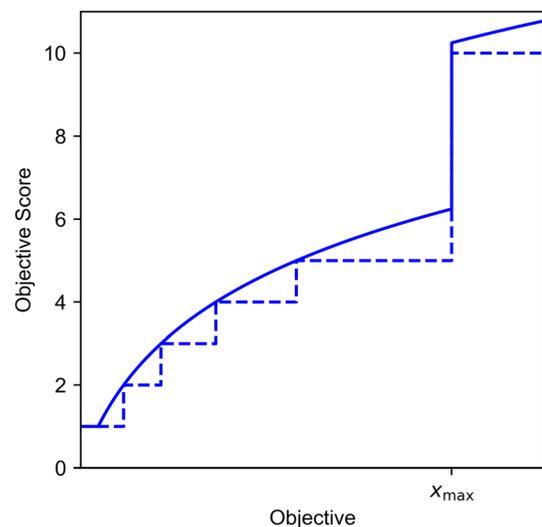

Fig. 3 This figure shows how we smoothed out the objective scoring for the FMECA utility. The dashed line is the normal, binned scoring from the original FMECA, and the solid line is the scoring we used to improve the training of MO-DCMAC. The increase from 5 to 10 is done at x_{max}

setting. Lastly, other methods for utility-based maintenance optimization, such as genetic algorithms [8, 11, 15], cannot scale to problem instances of this size.

6.4 Hyperparameters

In Table 1, we state the hyperparameters we used for our experiments. The hyperparameters were found by manual search. Most importantly, we only adjusted the hyperparameters between the utility function, and only γ, $V_{\min}$, and $V_{\max}$. This entails that we mostly use the same hyperparameters between the environments. The network architecture is a feedforward neural network. The actor network has one shared input layer. Each head has a hidden layer of size 50 and an output layer. The critic network consists of one hidden layer with 150 neurons.

For all the environments, we set the cost as the percentual value of replacement. Therefore, if all the components are replaced simultaneously, the cost for that time step would be 1. We found that during the initial testing, this significantly improved the accuracy because the original value for the cost of components was in the thousands, and higher values are harder to learn for a neural network. Another benefit is that this allowed hyperparameters to be shared between the environments. Otherwise, the $V_{\min}$ for cost must differ between environments, which could also affect other hyperparameters. Lastly, an episode consists of 50 years, each timestep representing a single year.

6.5 Software and hardware

We used Python 3.10 for our implementation. We used PyTorch 2.1, and Gymnasium 0.28 for the environments. The experiments were conducted on an Apple M1 Pro CPU with 16GB RAM.

7 Results and discussion

In Table 2, we show the results of each environment for both utility functions, while Fig. 4 shows the training plots with the FMECA utility and Fig. 5 for the threshold utility. In summary, MO-DCMAC outperforms the baseline policies in almost all the experiments, with the only exception being the larger quay wall asset with the threshold utility, which we will explain later in this section. Moreover, the YBA policies perform significantly worse than MO-DCMAC and the other baseline policies due to them being too passive and not immediately reacting to failing components. When we compare the inspection-based policies, YBI-CBA and CBI-CBA, to each other, we notice that CBI-CBA consistently performs better than YBI-CBA,

with the results only comparable at the simple homogenous asset.

For the simple homogenous asset environment, our results indicate that MO-DCMAC achieves the best utility score with both utility functions. When we compare the results of both objectives, cost, and probability of collapse, we notice that MO-DCMAC achieves a significantly lower cost (FMECA: 1.471 ± 0.323, Threshold: 1.498 ± 0.325) than the inspection-based policies YBI-CBA (FMECA: 1.636 ± 0.319, Threshold: 1.651 ± 0.304) and CBI-CBA (FMECA: 1.623 ± 0.338, Threshold: 1.615 ± 0.339), with approximately a 10% difference for both utility functions. In turn, the probability of collapse is slightly higher with MO-DCMAC (FMECA: 0.059 ± 0.052, Threshold: 0.058 ± 0.051) compared to CBI-CBA (FMECA: 0.056 ± 0.052, Threshold: 0.054 ± 0.052) and YBI-CBA (FMECA: 0.058 ± 0.061, Threshold: 0.053 ± 0.057). However, this difference is almost insignificant. This indicates that MO-DCMAC can learn how to balance cost versus probability of collapse by not performing costly maintenance if it only decreases the probability of collapse by a small amount. Another interesting result with the simple homogenous asset environment is that the difference in results between MO-DCMAC with the threshold utility and the FMECA utility is insignificant.

In Table 2, we see that the scores of both objectives, cost and probability of collapse, are almost identical, with MO-DCMAC with the FMECA utility getting 1.471 ± 0.323 for cost, and with the threshold getting 1.498 ± 0.325, while for the probability of collapse, MO-DCMAC with the FMECA utility is 0.059 ± 0.052 and with the threshold 0.058 ± 0.051. The most likely cause is that due to this environment being less complex, with no dependencies between the components, fewer components, and all the components of the same type, there is less flexibility in finding an optimized maintenance plan for this kind of asset. Therefore, it is likely that other utility functions, if correctly formulated, should not lead to different maintenance strategies. This, unfortunately, invalidates one benefit of MO-DCMAC for this environment over other methods, namely adjusting the utility function should result in different maintenance plans.

With the second environment, the quay wall asset, we see in Table 2 that MO-DCMAC again achieves the best utility score with both utilities. Moreover, the difference between MO-DCMAC (FMECA: 5.072 ± 3.158, Threshold: 1.133 ± 1.186) and CBI-CBA (FMECA: 5.566 ± 3.544, Threshold: 1.505 ± 0.492), the second best policy, is larger in this environment than in the simple homogenous asset environment. For instance, the difference in utility score with the quay wall asset environment is 9.34% with the FMECA utility and 33.42% with the threshold utility, whereas with the simple homogenous asset

Table 1 Hyperparameters used for the experiments

Utility	Threshold	FMECA
Training Steps	2.5×10^7	2.5×10^7
Actor LR	(2E−4, 2E−5)[1]	(2E−4, 2E−5)[1]
Critic LR	(2E−3, 2E−4)[1]	(2E−3, 2E−4)[1]
Activation	Tanh	Tanh
Clip Grad Norm	100	100
Update Every	128	128
$\mathcal{N}$	11	11
λ_{val}	0.5	0.5
λ_{ent}	0.001	0.001
γ	0.995	0.975
V_{min} Cost	− 12	− 4
V_{max} Cost	0	0
V_{min} Log Probability of Collapse	− 0.1	− 0.02
V_{max} Log Probability of Collapse	0	0

[1] The learning rate is linearly annealed during training from the start value to the end value

environment, this difference is only 1.64% with the FMECA utility and 6.94% with the threshold utility. This difference shows that MO-DCMAC can outperform other policies by a larger margin when the maintenance problem becomes more complex because it can likely learn the dependencies between components.

When we focus on the results of each utility, we see that MO-DCMAC achieves the best utility score and cost for both utilities. We see that the difference in cost between MO-DCMAC (FMECA: 1.028 ± 0.156, Threshold: 0.931 ± 0.15) and CBI-CBA (FMECA: 1.49 ± 0.227, Threshold: 1.479 ± 0.236) is substantial. Moreover, with the FMECA utility, the difference between the probability of collapse of MO-DCMAC (0.015 ± 0.02) and CBI-CBA (0.01 ± 0.021) is only 0.005, indicating that MO-DCMAC learns that lowering the probability of collapse after a specific value is less valuable for a better FMECA score.

The results of the threshold utility function for the quay wall asset environment show an even lower cost with MO-DCMAC (0.931 ± 0.15) compared to CBI-CBA (1.479 ± 0.236); however, the probability of collapse of MO-DCMAC (0.028 ± 0.033) is significantly higher than of CBI-CBA (0.01 ± 0.018). This difference is because MO-DCMAC learned it should not do certain maintenance actions if it decreases the probability of collapse beyond the threshold. Nevertheless, we see in Table 2 that the probability of collapse is, on average, always under the threshold of 0.1. This is true for all the environments, which indicates that during the training process, MO-DCMAC wants to learn a policy that can also handle unforeseen failures that can suddenly increase the probability of collapse. Still, it does not consistently achieve a

probability of collapse below the threshold, with the utility score always being higher than the cost. This effect is especially noticeable in the quay wall asset environment with MO-DCMAC (1.133 ± 1.186) because the standard deviation is almost equal to the mean utility score, while this is not the case with the other policies. Therefore, with MO-DCMAC, the probability of being above the threshold is higher than with other policies. A solution might be to increase the penalty if the probability of collapse is above the threshold.

In the last environment, the larger quay wall asset, we see in Table 2 that only the FMECA utility MO-DCMAC (7.293 ± 4.74) can outperform the other policies, such as CBI-CBA (10.806 ± 7.183). The difference in utility score with FMECA between MO-DCMAC and CBI-CBA, the second best-performing policy with FMECA, is larger than with the other environments, with a difference of 46.165%. When we compare the results of the objectives, we notice that MO-DCMAC can achieve the lowest cost (1.44 ± 0.284) and the second lowest probability of collapse (0.022 ± 0.028).

In contrast, the results of the larger quay wall asset environment with the threshold utility show that MO-DCMAC had a significantly worse utility score (3.264 ± 4.114) than CBI-CBA (2.214 ± 2.168). This is the only instance where MO-DCMAC's utility score is worse than that of other policies. The most probable reason is that MO-DCMAC cannot converge to a stable policy with the threshold utility function for the larger quay wall environment. We can see in training plots in Fig. 5 that the standard deviation is still noticeable. When we compare this with the training plots of the larger quay wall asset

Table 2 Results of the experiments in which we note the mean and standard deviation of 5000 episodes

Environment	Utility score	Policy	Score	Cost	Probability of collapse
Simple Homogenous Asset	FMECA	YBA-Repair	36.104 ± 5.74	**0.75 ± 0**	0.747 ± 0.224
		YBA-Replace	41.394 ± 41.637	10 ± 0	0.067 ± 0.098
		CBI-CBA	13.271 ± 7.777	1.623 ± 0.338	**0.056 ± 0.052**
		YBI-CBA	13.53 ± 9.339	1.636 ± 0.319	0.058 ± 0.061
		MO-DCMAC	**13.057 ± 7.419**	1.471 ± 0.323	0.059 ± 0.052
	Threshold	YBA-Repair	13.258 ± 2.412	**0.75 ± 0**	0.749 ± 0.23
		YBA-Replace	17.355 ± 13.878	5 ± 0	0.158 ± 0.177
		CBI-CBA	2.451 ± 2.467	1.615 ± 0.339	0.054 ± 0.052
		YBI-CBA	2.58 ± 3.181	1.651 ± 0.304	**0.053 ± 0.057**
		MO-DCMAC	**2.292 ± 2.455**	1.498 ± 0.325	0.058 ± 0.051
Quay Wall Asset	FMECA	YBA-Repair	16.486 ± 15.198	2.5 ± 0	0.063 ± 0.096
		YBA-Replace	17.451 ± 14.79	10 ± 0	0.01 ± 0.031
		CBI-CBA	5.566 ± 3.544	1.49 ± 0.227	**0.01 ± 0.021**
		YBI-CBA	9.567 ± 9.54	1.48 ± 0.2	0.039 ± 0.08
		MO-DCMAC	**5.072 ± 3.158**	**1.028 ± 0.156**	0.015 ± 0.02
	Threshold	YBA-Repair	5.404 ± 6.073	2.5 ± 0	0.061 ± 0.089
		YBA-Replace	7.032 ± 6.534	5 ± 0	0.032 ± 0.075
		CBI-CBA	1.505 ± 0.492	1.479 ± 0.236	**0.01 ± 0.018**
		YBI-CBA	2.814 ± 3.888	1.489 ± 0.202	0.04 ± 0.082
		MO-DCMAC	**1.133 ± 1.186**	**0.931 ± 0.15**	0.028 ± 0.033
Larger Quay Wall Asset	FMECA	YBA-Repair	29.127 ± 20.424	2.5 ± 0	0.14 ± 0.137
		YBA-Replace	22.045 ± 21.016	10 ± 0	**0.02 ± 0.045**
		CBI-CBA	10.806 ± 7.183	1.453 ± 0.169	0.046 ± 0.052
		YBI-CBA	16.039 ± 13.19	1.541 ± 0.141	0.083 ± 0.108
		MO-DCMAC	**7.293 ± 4.74**	**1.44 ± 0.284**	0.022 ± 0.028
	Threshold	YBA-Repair	10.1 ± 8.523	2.5 ± 0	0.143 ± 0.139
		YBA-Replace	10.576 ± 10.413	6 ± 0	0.057 ± 0.1
		CBI-CBA	**2.214 ± 2.168**	**1.461 ± 0.169**	**0.044 ± 0.044**
		YBI-CBA	4.583 ± 5.52	1.536 ± 0.144	0.084 ± 0.111
		MO-DCMAC	3.264 ± 4.114	1.568 ± 0.183	0.059 ± 0.07

The score column contains the utility score of the corresponding utility function. The bold results are the best results for the environment and utility combination, whereby if the mean is equal, a lower standard deviation is preferable. For each column, a lower score is preferable

environment with the FMECA utility in Fig. 4, we see that MO-DCMAC converges to a stable policy. If we compare the training plots in Figs. 4, 5 for the other environment, we see that with the threshold utility, it takes significantly more training steps before MO-DCMAC converges to a stable policy, compared to FMECA. This difference can be attributed to the fact that with the FMECA utility, MO-DCMAC receives more signals about when it performs well because there is always a benefit to lowering the probability of collapse and cost, while with the threshold utility, it only does so with cost.

Another probable cause for the lower performance with the larger quay wall environment could be the neural network architecture used. The current network architecture is a feedforward network, with one shared input layer for all components and an output head for each component. Due to the increased number of components in this environment, the shared layer struggles with learning each interaction. Increasing the size of the shared input layer might alleviate this, but we do not believe this is the best solution. DDMAC [5] did not use a shared layer and used separate input layers for each component. This network setup is feasible for the simple homogenous asset environment. Still, it would likely not perform well for the other environments because each maintenance action is taken independently of the states of the other components, and the probability of collapse depends on other components in the same sub-dependency group.
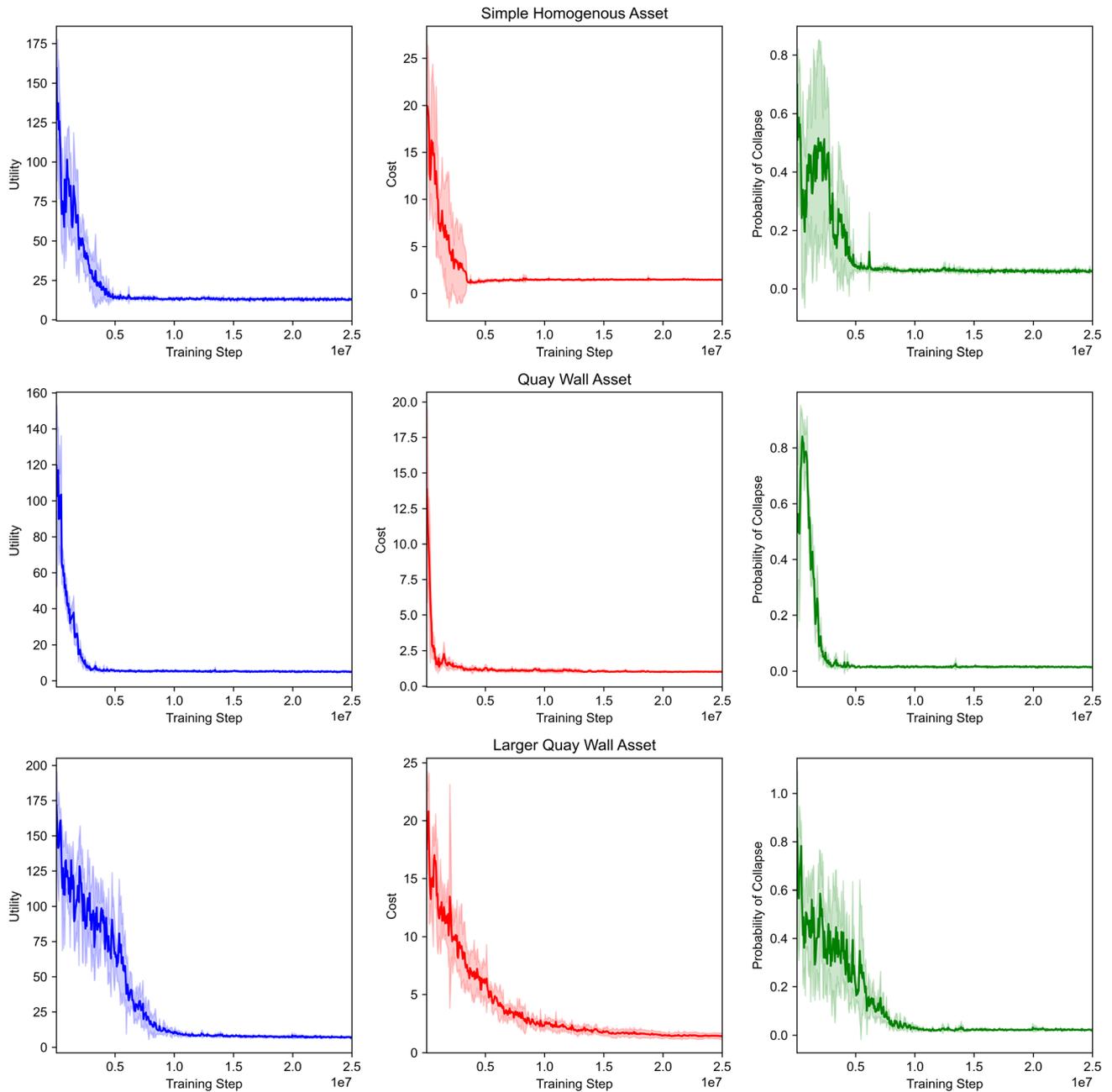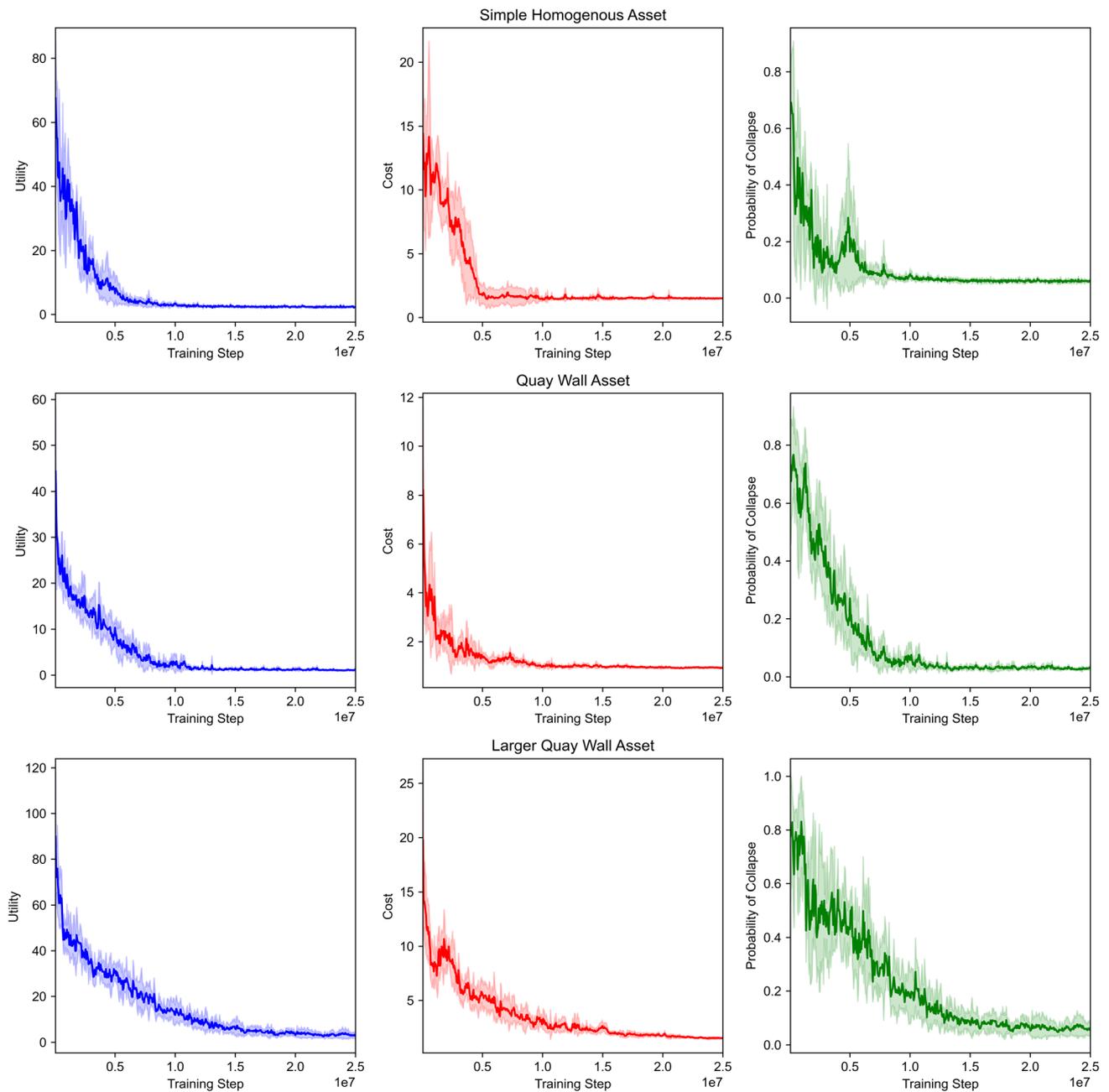
Fig. 4 Training plots of MO-DCMAC with the FMECA utility function. The line shows the mean over 5 runs, with the shaded area being the standard deviation

We believe the best solution would be to change the shared input layer to a graph neural network layer, such as GCN [23]. In it, the nodes would be the components, and we would connect nodes in the same sub-dependency group. Using a graph neural network layer would allow each component's action to consider other components in the same sub-dependency group but not the other components.

In Sect. 6.2.1, we stated how we wanted to show the effect on the threshold utility with different discount factors. We argued that a low discount factor might result in a sub-optimal policy due to the difference between the discounted and non-discounted returns, which can have an effect on the utility score. For these experiments, we used the simple homogenous asset environment with the following discount factors: $\gamma = 0.9$, $\gamma = 0.975$, $\gamma = 0.99$, $\gamma = 0.995$, and $\gamma = 1$

The results in Fig. 6 show how, with the threshold utility function, a higher discount factor positively correlates with the utility score. Moreover, the right plot in Fig. 6 signifies

Fig. 5 Training plots of MO-DCMAC with the threshold utility function. The line shows the mean over 5 runs, with the shaded area being the standard deviation

that with lower values of γ, the learned policy will result in a higher average probability of collapse. The primary reason for this is that due to the discounting, MO-DCMAC believes it has reached the threshold during training with greater ease and should not put more effort into lowering the probability of collapse. We see this in the middle plot of Fig. 6, where we see that increasing γ results in higher cost, with $\gamma = 0.9$ being the exception, likely due to this value of γ not being optimal for cost.

These experiments show the importance of selecting a correct γ based on the utility function when using MO-DCMAC. Especially if the utility function contains a strict threshold, meaning either above or below a specific objective value, the utility score is changed and not continuously. Another option would be to use a constrained MDP [14], where the threshold objective is a constraint that should be satisfied. This approach has other benefits when combined with either MO-DCMAC or MOCAC, namely the inability of these methods to scale to larger sets of

objectives due to the output of the critic scaling exponentially to the number of objectives. Therefore, if MO-DCMAC is combined with constrained MDP, we could model the probability of collapse as a constraint instead of an objective. This would allow other objectives in its place, such as the availability of the asset. Moreover, other research shows the application of constrained MDP in infrastructural maintenance optimization [5].

8 Conclusion

In this paper, we introduced MO-DCMAC, a utility-based MORL approach that can learn to plan maintenance for a multi-component asset. Our results show how MO-DCMAC can outperform commonly used heuristics for maintenance optimization in multiple environments.

We showed that a utility-based MORL approach for infrastructural maintenance optimization shows promise. We show that MO-DCMAC can learn a policy with a utility based on the FMECA methodology, which is currently used by asset managers to asses maintenance plans. This will allow the learned policy to match the requirements set by those asset managers closely. Moreover, MO-DCMAC can learn different policies depending on the utility used. This is essential because asset managers might need to adjust the utility function because of stronger constraints on the budget or different safety requirements.

MO-DCMAC was able to show these results in maintenance environments with real-world assets based on deterioration models and interaction between components. However, for future research, we might consider using surrogate models as a basis for a maintenance optimization environment [2]. These surrogate models can more accurately model the infrastructural assets; therefore, the resulting maintenance policy learned by MO-DCMAC will also be more accurate when implemented in the real world. Besides maintenance optimization, we believe that MO-DCMAC can also be used for other multi-objective problems requiring a factorized action space with a known utility function.

In this paper, we only considered the known utility function scenario described by Hayes et al. [18]. However, adapting MO-DCMAC to the dynamic utility function scenario has major benefits because asset managers or policymakers might adjust the thresholds in the threshold utility or the weighting in the FMECA utility due to new safety requirements or budget constraints. Having MO-DCMAC be able to learn an optimal policy for these dynamic priorities might be more desired than our current approach. Nevertheless, it is essential that the objectives can still be weighted using a nonlinear utility under the ESR criterion in this scenario.

MO-DCMAC's major limitation is its inability to scale to more objectives. The number of objectives is due to how the critic's output scales exponentially with the number of objectives. This limited us to using only two objectives in this paper; however, if we look at how maintenance is planned, we see that in most instances, significantly more objectives are considered, such as the availability of the structure. Therefore, the ability to scale up the number of objectives would increase the likelihood that these methods, such as MO-DCMAC, can be used in the real world drastically. We must investigate other approaches to accommodate more objectives [19].
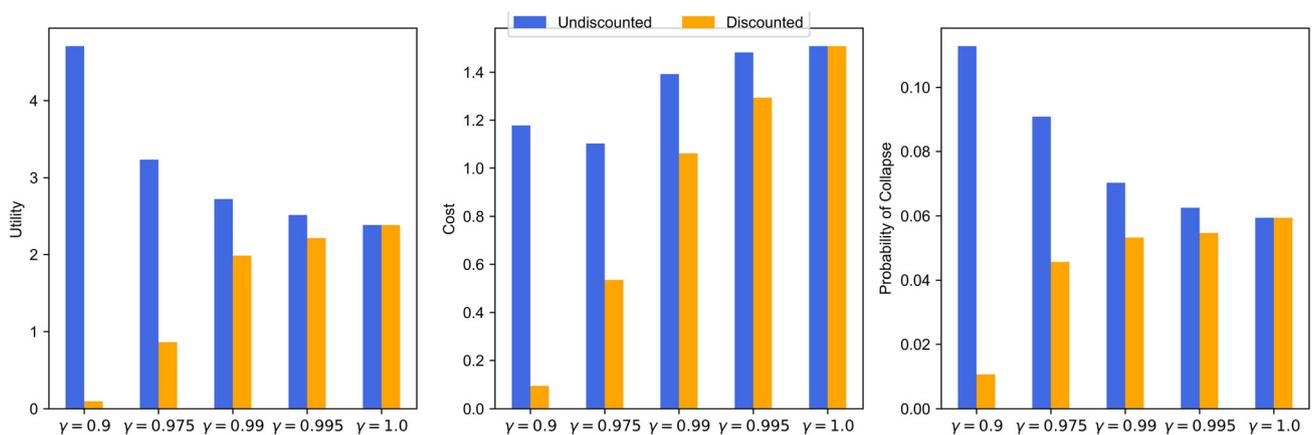

Fig. 6 This figure shows the effect of using a different γ with the threshold utility function with the simple homogenous asset environment. The blue bar is the non-discounted value, and the orange bar is the discounted value, which is the value if we apply the corresponding γ. For each of the plots, a lower score is always strictly better

Appendix 1: Environments

In this appendix, we state the details of each environment we used in our experiments. In each subsection, we state the components, their degradation matrix, the cost of each action, and the probability of collapse when a component or a group of components fails. In this first section, we note the implementation details shared between each environment.

In each environment, the state space of the components consists of the health state, which can be in one of five states, and the degradation rate τ, which starts at 0 and with $\tau_{max} = 50$. If a component c_i is in the last state, it is considered failing.

Each environment has the same available actions for both the global actions and the component actions. The global actions are as follows: *do nothing* and *inspect*. The component actions are: *do nothing*, *repair*, and *replace*. If a *repair* action is taken, the health state is improved with one step, and the current degradation rate does not change. With a *replace* action, the component is set to the best health state, and the degradation rate is reset at the next time step $\tau_{t+1}^{c_i} = 0$. Lastly, if a *do nothing* action is done for the component, the component health state can stay the same or transition to a worse.

This transition probability is described in a degradation matrix $D \in \mathbb{R}^{K \times \tau_{max} \times 5 \times 5}$, where K is the different number of component groups and τ_{max} the maximum degradation rate. For the degradation matrix of each component group $k \in K$, we only state the initial degradation matrix when $\tau = 0$ and the degradation matrix when $\tau = \tau_{max}$. The other degradation matrices are interpolated through the following formula:

$$
\begin{aligned}
D_{k,\tau=i} = D_{k,\tau=0} + \frac{i}{\tau_{max} - 1} \\
\cdot \left(D_{k,\tau=\tau_{max}} - D_{k,\tau=0} \right) \\
\forall k \in K \wedge \forall i \in \{0, \ldots, \tau_{max} - 1\}
\end{aligned}
\tag{27}
$$

Each environment also has the same observation space, whereby both the global and component actions influence the observation received. If both the global action and the component action for c_i are *do nothing*, the observation is as follows for component c_i:

$$
O^{donothing} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.
\tag{28}
$$

However, if either the global action is *inspect* or the

component action for c_i is *repair* or *replace*, the observation will fully reveal the state and is therefore:

$$
O^{reveal} = I_5.
\tag{29}
$$

Lastly, at the start of the episode, the belief state of each component will be set to:

$$
\mathbf{b}_0 = \{0.2, 0.2, 0.2, 0.2, 0.2\}
\tag{30}
$$

Simple Homogeneous Asset

The simple homogeneous asset environment consists of $N = 8$ components. Each component c_i can either be in one of five states, with each state being progressively worse than the previous state. When the component is in the last state, the component is failing. At each time step, a component c_i can transition to a worse state. This transition probability is stated in the following degradation matrices, with the initial degradation matrix being:

$$
D_{1,\tau=0} = \begin{bmatrix} 0.97 & 0.015 & 0.01 & 0.004 & 0.001 \\ 0 & 0.98 & 0.012 & 0.005 & 0.003 \\ 0 & 0 & 0.981 & 0.018 & 0.01 \\ 0 & 0 & 0 & 0.985 & 0.015 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},
\tag{31}
$$

and the final degradation matrix being:

$$
D_{1,\tau=\tau_{max}} = \begin{bmatrix} 0.9 & 0.05 & 0.03 & 0.015 & 0.005 \\ 0 & 0.91 & 0.06 & 0.02 & 0.01 \\ 0 & 0 & 0.92 & 0.06 & 0.02 \\ 0 & 0 & 0 & 0.93 & 0.07 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},
\tag{32}
$$

The degradation matrices for $0 < \tau < \tau_{max}$ are calculated by Eq. 27. At the start of the episode, each component starts at its best health state.

For each components c_i, we have three actions, namely *do nothing*, *repair*, and *replace*. Each has the following costs: 0, 0.0125, and 0.03125, respectively. There are also two global actions, namely *do nothing* and *inspect*, whereby the costs are 0 and 0.02.

Lastly, there is no interaction between components when the probability of collapse is calculated because each component belongs to an individual component group. Therefore, if a single component fails, the contribution to the probability of collapse is 0.05.

Quay Wall Asset

The second environment models a simplified version of a section of a historical quay wall in Amsterdam. In it, the goal is to plan maintenance for the wooden components of

a section of a quay wall. The environment consists of 13 components and three different component types. The first group is the wooden poles, which are comprised of nine poles, poles $= \{c_1, \ldots c_9\}$. The degradation matrices for the poles are:

$$D_{\text{pole},\tau=0} = \begin{bmatrix} 0.983 & 0.0089 & 0.0055 & 0.0025 & 0.0001 \\ 0 & 0.9836 & 0.0084 & 0.0054 & 0.0026 \\ 0 & 0 & 0.9862 & 0.0084 & 0.0054 \\ 0 & 0 & 0 & 0.9917 & 0.0083 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (33)$$

and

$$D_{\text{pole},\tau=\tau_{\max}} = \begin{bmatrix} 0.9713 & 0.0148 & 0.0093 & 0.0045 & 0.0001 \\ 0 & 0.9719 & 0.0142 & 0.0093 & 0.0046 \\ 0 & 0 & 0.9753 & 0.0153 & 0.0094 \\ 0 & 0 & 0 & 0.9858 & 0.0142 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (34)$$

The second group are the three wooden kesps, kesps $= \{c_{10}, c_{11}, c_{12}\}$, which have the following degradation matrices:

$$D_{\text{kesp},\tau=0} = \begin{bmatrix} 0.9748 & 0.013 & 0.0081 & 0.004 & 0.0001 \\ 0 & 0.9754 & 0.0124 & 0.0081 & 0.0041 \\ 0 & 0 & 0.9793 & 0.0125 & 0.0082 \\ 0 & 0 & 0 & 0.9876 & 0.0124 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (35)$$

and

$$D_{\text{kesp},\tau=\tau_{\max}} = \begin{bmatrix} 0.9534 & 0.0237 & 0.0153 & 0.0075 & 0.0001 \\ 0 & 0.954 & 0.0231 & 0.0152 & 0.0077 \\ 0 & 0 & 0.9613 & 0.0233 & 0.0154 \\ 0 & 0 & 0 & 0.9767 & 0.0233 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (36)$$

The last component, c_{13}, is the wooden floor, for which the degradation matrices are:

Table 3 Cost of each maintenance action in the quay wall environment

Component	Do nothing	Repair	Replace
Pole	0	0.011	0.044
Kesp	0	0.003	0.013
Floor	0	0.028	0.113

$$D_{\text{floor},\tau=0} = \begin{bmatrix} 0.9848 & 0.008 & 0.0049 & 0.0022 & 0.0001 \\ 0 & 0.9854 & 0.0074 & 0.0048 & 0.0024 \\ 0 & 0 & 0.9876 & 0.0075 & 0.0049 \\ 0 & 0 & 0 & 0.9926 & 0.0074 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (37)$$

and

$$D_{\text{floor},\tau=\tau_{\max}} = \begin{bmatrix} 0.9748 & 0.013 & 0.0081 & 0.004 & 0.0001 \\ 0 & 0.9754 & 0.0124 & 0.0081 & 0.0041 \\ 0 & 0 & 0.9793 & 0.0125 & 0.0082 \\ 0 & 0 & 0 & 0.9876 & 0.0124 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (38)$$

The cost of each maintenance action can be found in Table 3. The cost of the global actions is 0 for *do nothing* and 0.02 for *inspection*.

In this environment, we do consider interaction with component groups. The first type of component groups is wooden poles that influence each other. We define the following groups of this type:

$$\begin{aligned} \mathcal{G}_0^{\text{poles}} &= \{c_0, c_1, c_2\} \\ \mathcal{G}_1^{\text{poles}} &= \{c_3, c_4, c_5\} \\ \mathcal{G}_2^{\text{poles}} &= \{c_6, c_7, c_8\}. \end{aligned} \quad (39)$$

When one or more component fails in these groups, the effect on the probability of collapse is found in Table 4.

The second sub-dependency groups are the kesps that affect each other. These are defined as follows:

Table 4 Effect on the probability of collapse if one or more poles fail in the same sub-dependency groups

Components fail	Probability of collapse
0	0
1	0.01
2	0.1
3	0.4

Table 5 Effect on the probability of collapse if one or more kesps fail in the same sub-dependency groups

Components fail	Probability of collapse
0	0
1	0.03
2	0.33

Table 6 Effect on the probability of collapse if the floor fails in the same sub-dependency groups

Components fail	Probability of collapse
0	0
1	0.05

$$\begin{aligned}\mathcal{G}_0^{\text{kesps}} &= \{c_9, c_{10}\}\\ \mathcal{G}_1^{\text{kesps}} &= \{c_{10}, c_{11}\}.\end{aligned} \tag{40}$$

The effect of the probability of collapse is defined in Table 5.

Lastly, we consider the wooden floor, which itself belongs to a single sub-dependency group:

$$\mathcal{G}_0^{\text{floor}} = \{c_{12}\}. \tag{41}$$

with the effect of it defined in Table 6.

Due to insufficient maintenance over the last years, we assume the components are in states 3 and 4 just before the failed state (state 5). The starting state of the components is as follows:

$$s_0 = \{4, 4, 3, 4, 3, 3, 4, 3, 4, 3, 4, 3, 3\} \tag{42}$$

Larger Quay Wall Asset

The last environment, larger quay wall, is similar to the previous environment, quay wall, with the major difference being that the number of components is doubled to $N = 26$. Therefore, we have the same component groups of wooden poles, kesps, and floors. The wooden poles are $\{c_1, \ldots, c_{18}\}$, with the degradation matrices found in

Table 7 Cost of each maintenance action in the larger quay wall environment

Component	Do nothing	Repair	Replace
Pole	0	0.0055	0.022
Kesp	0	0.0015	0.0065
Floor	0	0.014	0.0565

Eqs. 33 and 34, the wooden kesps are $\{c_{19}, \ldots, c_{24}\}$ with the degradation matrices found in Eqs. 35 and 36, and the wooden floors are $\{c_{25}, c_{26}\}$ with the degradation matrices found in Eqs. 35 and 36. The cost of each maintenance action is found in Table 7.

The sub-dependency groups are as follows for the wooden poles:

$$\begin{aligned}\mathcal{G}_0^{\text{poles}} &= \{c_0, c_1, c_2\}\\ \mathcal{G}_1^{\text{poles}} &= \{c_3, c_4, c_5\}\\ \mathcal{G}_2^{\text{poles}} &= \{c_6, c_7, c_8\}\\ \mathcal{G}_3^{\text{poles}} &= \{c_9, c_{10}, c_{11}\}\\ \mathcal{G}_4^{\text{poles}} &= \{c_{12}, c_{13}, c_{14}\}\\ \mathcal{G}_5^{\text{poles}} &= \{c_{15}, c_{16}, c_{17}\}.\end{aligned} \tag{43}$$

The sub-dependency groups for the wooden kesps:

$$\begin{aligned}\mathcal{G}_0^{\text{kesps}} &= \{c_{18}, c_{19}\}\\ \mathcal{G}_1^{\text{kesps}} &= \{c_{19}, c_{20}\}\\ \mathcal{G}_2^{\text{kesps}} &= \{c_{20}, c_{21}\}\\ \mathcal{G}_3^{\text{kesps}} &= \{c_{21}, c_{22}\}\\ \mathcal{G}_4^{\text{kesps}} &= \{c_{22}, c_{23}\}.\end{aligned} \tag{44}$$

Lastly, the wooden floors are independent of each other; therefore, the sub-dependency groups are:

$$\begin{aligned}\mathcal{G}_0^{\text{floor}} &= \{c_{24}\}\\ \mathcal{G}_1^{\text{floor}} &= \{c_{25}\}.\end{aligned} \tag{45}$$

The effect of the probability of collapse is the same as the quay wall environment and can be found in Tables 4, 5, 6.

Similar to in the quay wall environment, we set the initial start state of the components to a worse state (State 3 or 4):

$$\begin{aligned}s_o = \{&4, 4, 3, 4, 3, 3, 4, 3, 4, 4, 4, 3, 4, 3,\\ &3, 4, 3, 4, 3, 4, 3, 3, 4, 3, 3, 4\}\end{aligned} \tag{46}$$

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Alegre LN, Bazzan AL, Roijers DM, et al (2023) Sample-efficient multi-objective learning via generalized policy improvement prioritization. In: Proceedings of the international joint conference on autonomous agents and multiagent systems, AAMAS, vol 2023-May. International foundation for autonomous agents and multiagent systems (IFAAMAS), pp 2003–2012. https://doi.org/10.5555/3545946.3598872

2. Aloisio A, Contento A, Alaggio R et al (2023) Physics-based models, surrogate models and experimental assessment of the vehicle-bridge interaction in braking conditions. Mech Syst Signal Process. https://doi.org/10.1016/j.ymssp.2023.110276

3. Andriotis. CP, Metwally. Z (2023) Structural integrity management via hierarchical resource allocation and continuous-control reinforcement learning. In: 14th international conference on applications of statistics and probability in civil engineering, ICASP14, 2262/103609

4. Andriotis CP, Papakonstantinou KG (2019) Managing engineering systems with large state and action spaces through deep reinforcement learning. Reliab Eng Syst Saf. https://doi.org/10.1016/j.ress.2019.04.036

5. Andriotis CP, Papakonstantinou KG (2021) Deep reinforcement learning driven inspection and maintenance planning under incomplete information and constraints. Reliab Eng Syst Saf. https://doi.org/10.1016/j.ress.2021.107551

6. Bellemare MG, Dabney W, Munos R (2017) A distributional perspective on reinforcement learning. In: 34th international conference on machine learning, ICML 2017

7. Bowles JB, Pel'aez CE (1995) Fuzzy logic prioritization of failures in a system failure mode, effects and criticality analysis. Reliab Eng Syst Saf. https://doi.org/10.1016/0951-8320(95)00068-D

8. Bukhsh ZA, Stipanovic I, Doree AG (2020) Multi-year maintenance planning framework using multi-attribute utility theory and genetic algorithms. Eur Transp Res Rev. https://doi.org/10.1186/s12544-019-0388-y

9. Chansombat S, Pongcharoen P (2018) Hicks C (2019) A mixed-integer linear programming model for integrated production and preventive maintenance scheduling in the capital goods industry. Int J Prod Res 10(1080/00207543):1459923

10. Correia MG, de Oliveira e. Bonates T, de Athayde Prata B, et al (2022) An integer linear programming approach for pavement maintenance and rehabilitation optimization. Int J Pavement Eng. https://doi.org/10.1080/10298436.2020.1869736

11. Dong Y, Frangopol DM, Saydam D (2014) Pre-earthquake multi-objective probabilistic retrofit optimization of bridge networks based on sustainability. J Bridge Eng 19:04014018. https://doi.org/10.1061/(asce)be.1943-5592.0000586

12. Dornheim J, Link N (2018) Multiobjective reinforcement learning for reconfigurable adaptive optimal control of manufacturing processes. In: 2018 13th international symposium on electronics and telecommunications, ISETC 2018—conference proceedings. https://doi.org/10.1109/ISETC.2018.8583854

13. Godsill S (2019) Particle filtering: The first 25 years and beyond. In: ICASSP, IEEE international conference on acoustics, speech and signal processing—proceedings. https://doi.org/10.1109/ICASSP.2019.8683411

14. Gu S, Yang L, Du Y, et al (2023) A review of safe reinforcement learning: methods, theory and applications. https://doi.org/10.48550/arXiv.2205.10330

15. Guan J, Yang X, You L et al (2022) Multi-objective optimization for sustainable road network maintenance under traffic equilibrium: incorporating costs and environmental impacts. J Clean Prod 334:130103

16. Hamida Z, Goulet JA (2023) Hierarchical reinforcement learning for transportation infrastructure maintenance planning. Reliab Eng Syst Saf 235:109214. https://doi.org/10.1016/j.ress.2023.109214

17. Hausknecht M, Stone P (2015) Deep recurrent q-learning for partially observable mdps. In: AAAI fall symposium—technical report

18. Hayes CF, Rădulescu R, Bargiacchi E et al (2022) A practical guide to multi-objective reinforcement learning and planning. Auton Agents Multi-Agent Syst 36:26. https://doi.org/10.1007/s10458-022-09552-y

19. Hayes CF, Verstraeten T, Roijers DM, et al (2022) Multi-objective coordination graphs for the expected scalarised returns with generative flow models. https://doi.org/10.48550/arXiv.2207.00368

20. de Jonge B, Teunter R, Tinga T (2017) The influence of practical factors on the benefits of condition-based maintenance over time-based maintenance. Reliab Eng Syst Saf 158:21–30. https://doi.org/10.1016/j.ress.2016.10.002

21. Kerkkamp D, Bukhsh ZA, Zhang Y, et al (2022) Grouping of maintenance actions with deep reinforcement learning and graph convolutional networks. In: International conference on agents and artificial intelligence. https://doi.org/10.5220/0010907500003116

22. Khodarahmi M, Maihami V (2023) A review on kalman filter models. Arch Comput Methods Eng 30:727. https://doi.org/10.1007/s11831-022-09815-7

23. Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: 5th international conference on learning representations, ICLR 2017—conference track proceedings

24. Korff M, Hemel MJ, Peters DJ (2022) Collapse of the grimburgwal, a historic quay in amsterdam, the netherlands. Proc Inst Civil Eng Forensic Eng 175:96. https://doi.org/10.1680/jfoen.21.00018

25. Lathourakis C, Andriotis CP, Cicirello A (2023) Inference and maintenance planning of monitored structures through markov chain monte carlo and deep reinforcement learning. In: 14th international conference on applications of statistics and probability in civil engineering, ICASP14, 2262/103339

26. Lavee D, Beniad G, Solomon C (2011) The effect of investment in transportation infrastructure on the debt-to-gdp ratio. Transp Rev 31:769. https://doi.org/10.1080/01441647.2011.596581

27. Martins LEG, Gorschek T (2016) Requirements engineering for safety-critical systems: a systematic literature review. Inf Softw Technol 75:71–89. https://doi.org/10.1016/j.infsof.2016.04.002

28. Méndez-Hernández BM, Rodríguez-Bazan ED, Martinez-Jimenez Y, et al (2019) A multi-objective reinforcement learning algorithm for jssp. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and

lecture notes in bioinformatics). https://doi.org/10.1007/978-3-030-30487-4_44

29. Milillo P, Giardina G, Perissin D et al (2019) Pre-collapse space geodetic observations of critical infrastructure: the morandi bridge, genoa, italy. Remote Sens 11:1403. https://doi.org/10.3390/rs11121403

30. Mnih V, Kavukcuoglu K, Silver D et al (2015) Human-level control through deep reinforcement learning. Nature 518:529. https://doi.org/10.1038/nature14236

31. Mnih V, Badia AP, Mirza L, et al (2016) Asynchronous methods for deep reinforcement learning. In: 33rd international conference on machine learning, ICML

32. Morato PG, Andriotis CP, Papakonstantinou KG et al (2023) Inference and dynamic decision-making for deteriorating systems with probabilistic dependencies through bayesian networks and deep reinforcement learning. Reliab Eng Syst Saf 235:109144. https://doi.org/10.1016/j.ress.2023.109144

33. Reymond M, Hayes CF, Steckelmacher D et al (2023) Actor-critic multi-objective reinforcement learning for non-linear utility functions. Auton Agents Multi-Agent Syst 37:23. https://doi.org/10.1007/s10458-023-09604-x

34. Roijers DM, Vamplew P, Whiteson S et al (2013) A survey of multi-objective sequential decision-making. J Artif Intell Res 48:67. https://doi.org/10.1613/jair.3987

35. Roijers DM, Whiteson S, Oliehoek FA (2015) Point-based planning for multi-objective pomdps. In: IJCAI international joint conference on artificial intelligence

36. Rădulescu R, Mannion P, Roijers DM et al (2020) Multi-objective multi-agent decision making: a utility-based analysis and survey. Auton Agents Multi-Agent Syst. https://doi.org/10.1007/s10458-019-09433-x

37. Silver D, Veness J (2010) Monte-carlo planning in large pomdps. In: advances in neural information processing systems 23: 24th annual conference on neural information processing systems 2010, NIPS

38. Skordilis E, Moghaddass R (2020) A deep reinforcement learning approach for real-time sensor-driven decision making and predictive analytics. Comput Ind Eng 147:106600. https://doi.org/10.1016/j.cie.2020.106600

39. Soh H, Demiris Y (2011) Evolving policies for multi-reward partially observable markov decision processes (mr-pomdps). In: Genetic and evolutionary computation conference, GECCO'11. https://doi.org/10.1145/2001576.2001674

40. Swanson L (2001) Linking maintenance strategies to performance. Int J Prod Econ 70:237. https://doi.org/10.1016/S0925-5273(00)00067-0

41. Watkins CJ, Dayan P (1992) Technical note: Q-learning. Mach Learn 8:279. https://doi.org/10.1023/A:1022676722315

42. Wawrzyński P (2009) Real-time reinforcement learning by sequential actor-critics and experience replay. Neural Netw 22:1484. https://doi.org/10.1016/j.neunet.2009.05.011

43. Yang L, Li G, Zhang Z et al (2021) Operations & maintenance optimization of wind turbines integrating wind and aging information. IEEE Trans Sustain Energy 12:211. https://doi.org/10.1109/TSTE.2020.2986586

44. Yang R, Sun X, Narasimhan K (2019) A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In: Advances in neural information processing systems

45. Zonta T, da Costa CA, da Rosa Righi R et al (2020) Predictive maintenance in the industry 4.0: s systematic literature review. Comput Ind Eng 150:106889. https://doi.org/10.1016/j.cie.2020.106889