A Deep Multi-Objective Reinforcement Learning Approach for Infrastructural Maintenance Planning with Non-Linear Utility Functions

Jesse van Remmerden[a;*], Maurice Kenter[b], Diederik Roijers[b,c], Yingqian Zhang[a], Charalampos Andriotis[d] and Zahara Bukhsh[a]

[a]Dept. of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology
[b]City of Amsterdam
[c]AI lab, Vrije Universiteit Brussel
[d]Faculty of Architecture and the Built Environment, Delft University of Technology
ORCiD ID: Jesse van Remmerden https://orcid.org/0009-0005-1966-6907,
Maurice Kenter https://orcid.org/0000-0001-7834-089, Diederik Roijers https://orcid.org/0000-0002-2825-2491,
Yingqian Zhang https://orcid.org/0000-0002-5073-0787,
Charalampos Andriotis https://orcid.org/0000-0002-0140-5021,
Zaharah Bukhsh https://orcid.org/0000-0003-3037-8998

Abstract. This paper explores the potential of multi-objective reinforcement learning (MORL) in infrastructural maintenance planning, an area traditionally dominated by single-objective RL approaches. In this paper, we introduce the Multi-Objective Deep Centralized Multi-Agent Actor Critic (MODCMAC), a MORL method for maintenance planning capable of optimizing a policy with a known non-linear utility function under the Expected Scalarized Return (ESR) criterion, while the state is only partially observable. Previous single-objective RL methods had to combine multiple objectives, such as risk and cost, into a singular reward signal through reward-shaping. In contrast, MODCMAC can optimize a policy for multiple objectives directly, even when the utility function is non-linear. We evaluated MODCMAC using a utility function based on the Failure Mode, Effects, and Criticality Analysis (FMECA) methodology, which used the failure probability and cost as input. The evaluation was done within an environment requiring optimizing a maintenance plan for a historical quay wall. The performance of MODCMAC was compared against a Belief-State-Based (BSB) policy with deterministic or stochastic action selection. Our findings indicate that MODCMAC outperforms the BSB policy. The code can be found at https://github.com/jesserem/MODCMAC.

1 Introduction

In September 2020, a quay wall collapsed along the Grimburgwal in the center of Amsterdam [12]. The quay wall, dating from 1870, consisted of many large wooden components and was at its time of collapse 150 years old. While old, it was not the oldest quay wall in the city, signifying that other factors contributed to its collapse besides age. Because of these many factors, maintaining quay walls is a difficult and essential task for the city, and much work is currently being done to maintain the quay walls.[1]

One strategy for performing maintenance on large assets such as quay walls is to define a proactive/reactive maintenance policy. This would entail pre-scheduling inspections and maintenance while also responding to signals (such as deformation measurements from satellites) with inspections and maintenance. This approach has two key issues: incidents occur at the expense of safety, and no accurate insight is obtained into the structural condition of the quay wall, leading to inefficient maintenance planning. Meanwhile, the capacity for inspections and maintenance is scarce, and it is expensive to perform inspections and maintenance. In this paper, we work towards a different approach to more effectively schedule the scarce and costly inspections and maintenance: prescriptive maintenance based on a decision-theoretic model [19]. This type of modeling has been shown to be capable of reducing costs in the maintenance of other large assets [2], and is, therefore, a promising research direction for quay wall maintenance.

A key insight behind decision-theoretic prescriptive maintenance is to explicitly take uncertainty about the condition state of the asset into account while deciding the best course of action. This is especially important in the case of quay walls, where, due to their very nature, a large part of them is located underwater and under a street. It is, therefore, difficult to fully observe the state of a quay wall's components and detect failed components. Furthermore, the state of the wooden components has been known to show a large variance across different quay walls. Taking this uncertainty into account is essential in forming an effective maintenance strategy.

A complicated but essential factor when performing maintenance on quay walls is how the prioritization of maintenance tasks is performed in practice. Specifically, this is not done based on one scalar reward signal (such as costs), as is typical in decision-theoretic mod-

* Corresponding Author. Email: j.v.remmerden@tue.nl

[1] For an overview of what is currently being done in terms of maintenance, see https://www.amsterdam.nl/projecten/kademuren/ (in Dutch).

els. This is, for example, reflected in the *Failure Mode, Effects and Criticality Analysis (FMECA)* methodology, where the utility (criticality score) of the different ways an asset can fail depends on both costs and risk (as well as other factors such as environment effects), which are combined in a non-linear manner. It is, therefore, key that such non-linear considerations can be optimized using a decision-theoretic approach as well.

In this paper, we formulate a *multi-objective partially observable Markov decision process model (MOPOMDP)* for the maintenance planning of quay walls. This MODPOMDP is formulated not only to explicitly handle uncertainty about the state of the components of the quay wall but also to model different reward signals that can be taken into account in a non-linear fashion. To do so, we propose a new method that we call *Multi-Objective Deep Centralized Multi-Agent Actor-Critic (MODCMAC)*, which is based on the DMCAC [2] and MOCAC [14] algorithms for maintenance planning and multi-objective reinforcement learning respectively. We show experimentally that this method can effectively optimize for a non-linear utility function inspired by the FMECA methodology. We further note that our method can be used to incorporate all relevant objectives that an asset manager would need to take into account.

2 Background

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 $\mathcal{M}_{\text{MOMDP}} = \left\langle S, A, T, \vec{R}, \gamma \right\rangle$, in which S and A describe the action and state space respectively. $T : S \times A \times S \rightarrow [0, 1]$ is the transition function. $\vec{R} : S \times A \times S \rightarrow \mathbb{R}^d$ is the reward function. This reward function returns a vector of size d, in which d is the number of objectives that need to be optimized.

Within MORL, we can either optimize for an unknown or a known utility function [9]. In our problem setting, we will have a known utility function. A utility function $u : \mathbb{R}^d \rightarrow \mathbb{R}$ is a mapping of the reward vector $\vec{r}_t$ to a singular value. It is essential for the utility function to be *monotonically increasing*, meaning that if one of the objectives increases, the utility return can never decrease. This seems to make any MO problem with a known utility function be able to learn by a single-objective RL method. However, this is not true when the utility function is non-linear.

SER and ESR criterion There are two methods of optimizing over a known non-linear utility function in MORL [9], namely the *Scalarized Expected Return* (SER) criterion, which is when the utility function is applied on the expected return:

$$\pi^* = \arg\max_{\pi} u\left(E\left[\sum_{t=0}^{H} \gamma^t \vec{r}_t | \pi, s_0\right]\right) \tag{1}$$

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

$$\pi^* = \arg\max_{\pi} E\left[u\left(\sum_{t=0}^{H} \gamma^t \vec{r}_t | \pi, s_0\right)\right] \tag{2}$$

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). For our approach, we will use the ESR criterion because we cannot afford to have a high failure probability in certain episodes, even though, on average, the probability is low. However, no standard single-objective RL method can optimize with this criterion due to the Bellman equation being invalid under those conditions as a non-linear utility function does not distribute over summations and expectations.

Partially Observable Markov Decision Process (POMDP) A common problem when applying RL to maintenance problems is that the true state of the structure is not fully known and can only be partially observed. Therefore, most maintenance environments are not formulated through an MDP but with a *Partially Observable Markov decision process* (POMDP) [17]. A POMDP is a tuple $\mathcal{M}_{\text{POMDP}} = \langle S, A, \Omega, T, O, R, \gamma \rangle$. One of the main differences between an MDP and a POMDP is that with a POMDP, the agent receives an inaccurate observation $o_t \in \Omega$ of the current state s_t instead of the state itself. The observation space Ω contains all the possible observations the agent can receive about the state space. Lastly, the observation function $O : S \times A \times \Omega \rightarrow [0, 1]$ returns an observation o_t, based on the current state s_t.

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 $|S|$. This belief is then updated at each timestep through a Bayesian Update:

$$\begin{aligned} b(s_{t+1}) &= p\left(s_{t+1}|o_{t+1}, a_t, \mathbf{b}_t\right) \\ &= \frac{p\left(o_{t+1}|s_{t+1}, a_t\right)}{p\left(o_{t+1}|\mathbf{b}_t, a_t\right)} \sum_{s_t \in S} p\left(s_{t+1}|s_t, a_t\right) b(s_t) \end{aligned} \tag{3}$$

Multi-Objective Partially Observable Markov Decision Process (MOPOMDP) In previous paragraphs, we discussed both the multi-objective Markov decision process (MOMDP) and the partially observable Markov decision process (POMDP). For our method, we will use a *multi-objective partially observable Markov decision process (MOPOMDP)* [16]. A MOPOMDP is a combination of the *partially observable Markov decision process (POMDP)* [17] and a *multi-objective Markov decision process (MOMDP)* [9], and is defined in definition 1.

Definition 1 *A **multi-objective partially observable Markov Decision Process** is a tuple $\mathcal{M}_{MOPOMDP} = \left\langle S, A, \Omega, T, O, \vec{R}, \gamma \right\rangle$. S is the state space, A is the action space, Ω is the observation space, $T : S \times A \times S \rightarrow [0, 1]$ is the transition function, $O : S \times A \times \Omega \rightarrow [0, 1]$ is the observation function, $\vec{R} : S \times A \times S \rightarrow \mathbb{R}^d$ is the reward function that returns a vector of d values for each objective, and $\gamma \in [0, 1]$ is the discount factor.*

In our method section, we will discuss how we formulated our maintenance problem as a MOPOMDP.

2.1 MOCAC

Previously, we discussed how no standard RL method could optimize with the ESR criterion if the utility function is non-linear due to the Bellman Equation being invalidated in this situation. MOCAC (Multi-Objective Categorical Actor-Critic) [14] can overcome this invalidation utilizing a distributional critic.

$$V(s_t) = \sum_i z_i p_i(s_t) \tag{4}$$

Equation 4 shows how the distributional critic functions by learning the distribution p of the future returns. The support vector z is a set of c bins, such that $z = \{z_i = V_{\text{MIN}} + i\Delta z : 0 \leq i \leq c\}$, with $\Delta z := \frac{V_{\text{MAX}} - V_{\text{MIN}}}{c-1}$. With V_{MIN} being the minimum expected return and V_{MAX} being the maximum.

However, the critic in MOCAC outputs a multivariate distribution instead of a univariate probability distribution. This is because the critic learns the probabilities of multiple objectives rather than just one. Therefore, a support atom $\vec{z}_i$ is not a singular value but a vectorial return of the objectives. Reymond et al. [14] modified how $\vec{z}$ is created by providing separate V_{MIN} and V_{MAX} for each objective while maintaining the same number of bins c for each objective. Each atom $\vec{z}_i$ is then calculated by:

$$\begin{aligned} &\vec{z}_i = \vec{V}_{\text{MIN}} + \Delta \vec{z}_i \\ &\text{with } \Delta \vec{z}_i := \frac{\vec{V}_{\text{MAX}} - \vec{V}_{\text{MIN}}}{c-1} \end{aligned} \tag{5}$$

Because each objective has c bins, the critic learns a discrete distribution of c^d. MOCAC uses this distribution with the accrued reward ($\vec{R}_t^- = \sum_{k=0}^{t-1} \gamma^k \vec{r}_k$) as input for the utility function u:

$$u_t = \sum_j u\Big(\vec{R}_t^- + \gamma^t \vec{z}_j\Big) p_j(s_t) \tag{6}$$

This allows Reymond et al. [14] to train the actor through the loss function found in equation 7. In it, Reymond et al. use the utility to calculate the advantage $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$, which helps determine how much better an action is compared to the other actions.

$$\begin{aligned} L(\pi) &= -\sum_{t=0}^{H} A(s_t, a_t) \log\left(\pi_\theta(a_t|s_t)\right) \\ &= -\sum_{t=0}^{H} (u\Big(\vec{R}_t^- + \gamma^t \left(\vec{r}_t + \gamma V_\psi(s_{t+1})\right)\Big) \\ &\quad - u\Big(\vec{R}_t^- + \gamma^t V_\psi(s_t)\Big)) \log\left(\pi_\theta(a_t|s_t)\right) \\ &= -\sum_{t=0}^{H} (\sum_j u\Big(\vec{R}_t^- + \gamma^t \left(\vec{r}_t + \gamma \vec{z}_j\right)\Big) p_j(s_{t+1}) \\ &\quad - \sum_j u\Big(\vec{R}_t^- + \gamma^t \vec{z}_j\Big) p_j(s_t)) \log\left(\pi_\theta(a_t|s_t)\right) \end{aligned} \tag{7}$$

Moreover, Reymond et al. [14] argue that conditioning on the accrued reward $\vec{R}_t^-$ by adding it to the state space is needed to get an optimal policy in environments with stochastic reward schemes.

2.2 *DCMAC*

A recent DRL approach that utilizes a POMDP and a Bayesian Belief for infrastructure maintenance is DCMAC (Deep Centralized Multi-Agent Actor Critic) [2]. DCMAC optimizes maintenance for a multi-component asset, in which, for each component, an individual action is taken. This would typically increase the size of the action space significantly. Therefore, Andriotis et al. use a factorized action space with a multi-head actor network. Due to the use of multi-head, a summation of all the log probabilities produced by heads has to be taken to calculate the policy loss. Furthermore, Andriotis et al. introduced truncated importance sampling $w_t = \min\left(c, \frac{\pi(a_t|s_t)}{\mu(a_t, s_t)}\right)$ to address the high variance that log summation may introduce. This resulted in the modified policy loss function, as shown below, with N being the number of components:

$$L(\pi) = -w_t \left(\sum_{j=1}^{N} \log\left(\pi_\theta^{(j)}\left(a_t^{(j)}|\mathbf{b}^t\right)\right) \right) A_t \tag{8}$$

3 Method

When using deep RL to plan a multi-component system's maintenance, we must address the curse of dimensionality of the action space. One solution to this curse of dimensionality is to use a factorized action space through DCMAC as proposed by Andriotis et al. [2]. However, DCMAC can optimize for one objective, while in real-world settings, most maintenance plans are made to optimize multiple objectives, such as maximizing the structure availability and minimizing the maintenance impact on the environment. A common approach is to translate these objectives to a cost, which is then combined in a single scalar reward value [2, 8, 11]. If one seeks to include methodologies that are used in maintenance planning, such as the FMECA methodology, extensive reward-shaping is needed through domain knowledge. We propose Multi-Objective Deep Centralized Multi-Agent Actor Critic (MODCMAC), a MORL method for maintenance planning, based on MOCAC [14] and DCMAC [2].

We use the definition of MOPOMDP (see definition 1) to formulate our maintenance problem, which is both multi-objective and partially observable:

- $s_t \in S$ The state-space is discrete, with the N components being able to be in one of $|S^{\text{comp}}|$ states and a degradation rate $\tau_t^{(i)}$ for each component.
- $a_t \in A$ The action-space is the combination of actions for the N component, which is $A^{\text{comp},i} \in \{\textit{nothing}, \textit{repair}, \textit{replace}\}$ with $1 \leq i \leq N$, and the global action for the whole structure, which is $A^{\text{global}} \in \{\textit{nothing}, \textit{inspect}\}$.
- $o_t \in \Omega$ The observation-space is equal to the state-space, except τ, in that the observation for each component can be one of $|S^{\text{comp}}|$ states.
- $T : S \times A \times S \to [0, 1]$ At each timestep t, the component has a probability of either staying in a specific state or transitioning to a worse state, based on τ_t 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 τ is set to 0.
- $O : S \times A \times \Omega \to [0, 1]$ The received observation is based on a_t^{global}. If the *nothing* action is taken, the observation will be of low accuracy representation of the state. If the *inspect* action is taken, the state will be fully observed.
- $\vec{R} : S \times A \times S \to \mathbb{R}^2$ The reward vector consists of two values: *cost* (r_c), which represent the monetary cost of taking certain actions, and *failure probability* (r_f) which represent the failure probability. We calculate r_f through $\log(1 - p_f)$, with p_f being the failure probability.

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 equation 3 [2].

In our MOPOMDP formulation, we stated that the *failure probability* reward is the log probability $r_f = \log(1 - p_f)$. This equation is based on a few key considerations. The first is that the utility function uses the episodic return as input, and we cannot add two independent calculated probabilities together. However, we know that

$\log(x*y) = \log(x) + \log(y)$, and we can return the log-probability to a normal probability through the exponent $p = e^{\log(p)}$. Moreover, we treat the failure probability at each timestep as an independent event. This means that we can add the probabilities through $p = 1 - \prod_{i=0}^{H}(1-p_i)$. The main issue with this is that in RL, the episodic return is a summation, but by using the log probability instead of the normal probability, we can add the failure probability from each timestep because:

$$1 - e^{\sum_i \log(1-p_i)} = 1 - \prod_i 1 - p_i \tag{9}$$

FMECA Utility Function Our utility function u evaluates the performance of MODCMAC by mapping the vectorial return to a preference score. We based our utility function on the *Failure Mode Effect and Criticality Analysis (FMECA)* methodology [4]. This FMECA methodology evaluates particular failure modes of an instance and the consequences when those failure modes occur. The goal of FMECA, in a maintenance planning setting, is to determine the most optimal maintenance plan 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; however, in our setting, we will focus only on the failure probability and the cost as the objectives.

The input of our utility function is the return $\vec{R}$ of the *cost* reward $R_c = \sum_t^H r_c$ and *failure probability* reward $R_f = \sum_t^H r_f$. We calculate the failure probability of the whole episode using R_f through:

$$p_f = 1 - e^{R_f} \tag{10}$$

Within the utility function, we have the following constants: The maximum allowed cost $C_{\max}$ and failure probability $F_{\max}$ before a penalty P is added. To see if the value of either return exceeds its maximum, we use the following formulas:

$$\text{pen}_{\text{cost}}(R_c) = \begin{cases} 1 & \text{if } R_c > C_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

$$\text{pen}_{\text{risk}}(p_f) = \begin{cases} 1 & \text{if } p_f > F_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

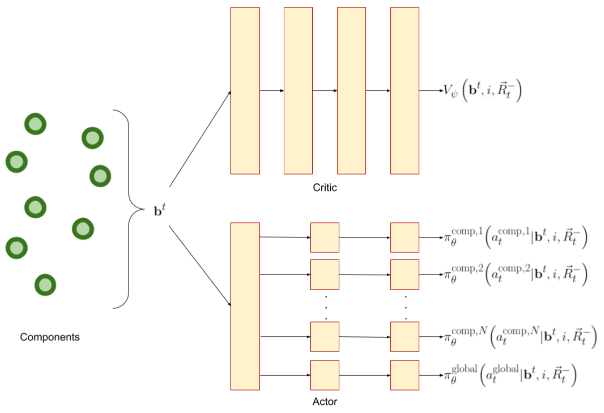
We then calculate the utility of the *cost* and the *failure probability* as:

$$c_{\text{utility}}(R_c) = 6 * \log_{10}\left(1 + \frac{R_c}{C_{\max}} * 10\right) + P * \text{pen}_{\text{cost}}(R_c) \tag{13}$$

$$f_{\text{utility}}(p_f) = 6 * \log_{10}\left(1 + \frac{p_f}{F_{\max}} * 10\right) + P * \text{pen}_{\text{risk}}(p_f) \tag{14}$$

One major adjustment that we made in the utility function, compared to the FMECA methodology, on which we base our utility function, is that we employed logarithm to base 10 ($\log_{10}$) Normally, in the FMECA methodology, 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 MODCMAC's training time. It may either trap the process 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.

$$u = -\left(\max\left(1, c_{\text{utility}}(R_c)\right) \times \max\left(1, f_{\text{utility}}(p_f)\right)\right) \tag{15}$$



$V_\psi\left(\mathbf{b}^t, i, \vec{R}_t^-\right)$
Critic
$\mathbf{b}^t$
Components
$\pi_\theta^{\text{comp},1}\left(a_t^{\text{comp},1} | \mathbf{b}^t, i, \vec{R}_t^-\right)$
$\pi_\theta^{\text{comp},2}\left(a_t^{\text{comp},2} | \mathbf{b}^t, i, \vec{R}_t^-\right)$
$\pi_\theta^{\text{comp},N}\left(a_t^{\text{comp},N} | \mathbf{b}^t, i, \vec{R}_t^-\right)$
$\pi_\theta^{\text{global}}\left(a_t^{\text{global}} | \mathbf{b}^t, i, \vec{R}_t^-\right)$
Actor

Figure 1: The network architecture of MODCMAC. The critic network is a four-layer network that outputs the multivariate distribution of the return. The actor network has one shared input layer and $N+1$ heads, where N is the number of components. Each of these heads has two layers, and the final layer outputs the action probabilities

Equation 15 calculates the last step of the utility. 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, MODCMAC would likely learn to either do no maintenance at all or maximize maintenance, such that the risk stays 0 because that would result in a utility score of 0.

3.1 Multi-Objective Deep Centralized Multi-Agent Actor-Critic

Our proposed method, *Multi-Objective Deep Centralized Multi-Agent Actor-Critic (MODCMAC)*, is based on DMCAC [2] and MOCAC [14], both actor-critic methods. Combining these methods would allow 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 non-linear utility function in an ESR setting by using the critic network of MOCAC. This section will focus on how we integrated both methods into MODCMAC.

The input for MODCMAC is the tuple $\left(\mathbf{b}^t, i, \vec{R}_t^-\right)$. The first value is the belief $\mathbf{b}^t$ at timestep t, which we update through equation 3, i is the normalized timestep $i = \frac{t}{H}$, with H being the horizon of the episode. We included it based on how DMCAC [2] uses it to signify the development of the deterioration rate τ. Lastly, we included the accrued reward $\vec{R}_t^-$ as necessitated by the muli-objective setting under the ESR with a non-linear utility [14].

Previously, we explained how the policy loss is calculated for DCMAC (equation 8). Within MODCMAC, we added an additional head for global actions of the structure $\pi^{\text{global}}\left(a_t^{\text{global}} | s_t\right)$. We adjusted equation 8 such that this global head is included in the loss calculation:

$$L(\pi) = -w_t\Big(\Big(\sum_{j=1}^{N} \log\left(\pi_\theta^{(j)}\left(a_t^{(j)} | \mathbf{b}^t\right)\right) + \log\left(\pi_\theta^{\text{global}}\left(a_t^{\text{global}} | \mathbf{b}^t\right)\right)\Big) A_t \tag{16}$$

The loss calculation in equation 16 is not modified for a multi-objective environment and does not include the normalized deterioration rate i and the accrued reward $\vec{R}_t^-$. Therefore, we combine this loss function with the one found in equation 7. This results in the policy loss for MODCMAC being:

$$\begin{aligned}L(\pi) = &-\sum_{t=0}^{H} w_t(u\Big(\vec{R}_t^- + \gamma^t\left(\vec{r}_t + \gamma V_\psi\left(\mathbf{b}_{t+1}, i_{t+1}, \vec{R}_{t+1}^-\right)\right)\Big)\\ &- u\Big(\vec{R}_t^- + \gamma^t V_\psi\left(\mathbf{b}_t, i_t, \vec{R}_t^-\right)\Big))\\ &*\Big(\Big(\sum_{j=1}^{N}\log\left(\pi_\theta^{(j)}\left(a_t^{(j)}|\mathbf{b}_t, i_t, \vec{R}_t^-\right)\right)\Big)\\ &+\log\left(\pi_\theta^{\text{global}}\left(a_t^{\text{global}}|\mathbf{b}_t, i_t, \vec{R}_t^-\right)\right)\Big)\end{aligned} \tag{17}$$

Our implementation of the critic network is essentially a replication of MOCAC, maintaining its core components, including the computation of loss and other significant attributes [14]. However, by utilizing this critic network in combination with the multi-head actor network approach of DCMAC, we had to make significant adjustments to the policy loss calculation (equation 17).

$$\text{Loss} = L(\pi) + c_{\text{value}}L(V) - c_{\text{entropy}}H(\pi) \tag{18}$$

Within DCMAC, the policy loss and the value loss are calculated and applied separately, and it uses a decaying epsilon-greedy strategy for exploring [2]. We combine both losses, similar to MOCAC [14] (equation 18). In it, Reymond et al. use the entropy and the entropy coefficient c_{entropy} for exploring, and the value coefficient c_{value}, which indicates the importance of the value loss. Tuning the value coefficient is more important if the critic and actor share layers.

4 Experiments and results

Environment Details We tested MODCMAC on a maintenance planning problem, which is based on previous research [2, 3]. We adjusted the simulation environment to model a simplified version of a section of a historical quay wall in Amsterdam. We model the agent to plan maintenance on the wooden components of a section of a quay wall. These components consist of nine poles $\{\text{comp}_1, ..., \text{comp}_9\}$, three pile cabs $\{\text{comp}_{10}, \text{comp}_{11}, \text{comp}_{12}\}$, and one floor $\{\text{comp}_{13}\}$. The total number of components is $N = 13$. For each type of component, we have two degradation matrices (as shown in equation 19 and 20 for the poles, equation 21 and 22 for the pile cabs, and equation 23 and 24 for the floor), one when the degradation rate $\tau = 0$ and one when the degradation rate is $\tau = 50$, which is the episode length.

$$D_{\tau=0}^{\text{pole}} = \begin{bmatrix} 0.983 & 0.0089 & 0.0055 & 0.0025 & 0.0001 \\ 0 & 0.9836 & 0.0084 & 0.0054 & 0.0026 \\ 0 & 0 & 0.9862 & 0.0084 & 0.0054 \\ 0 & 0 & 0 & 0.9917 & 0.0083 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{19}$$

$$D_{\tau=50}^{\text{pole}} = \begin{bmatrix} 0.9713 & 0.0148 & 0.0093 & 0.0045 & 0.0001 \\ 0 & 0.9719 & 0.0142 & 0.0093 & 0.0046 \\ 0 & 0 & 0.9753 & 0.0153 & 0.0094 \\ 0 & 0 & 0 & 0.9858 & 0.0142 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{20}$$

$$D_{\tau=0}^{\text{pile cab}} = \begin{bmatrix} 0.9748 & 0.013 & 0.0081 & 0.004 & 0.0001 \\ 0 & 0.9754 & 0.0124 & 0.0081 & 0.0041 \\ 0 & 0 & 0.9793 & 0.0125 & 0.0082 \\ 0 & 0 & 0 & 0.9876 & 0.0124 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{21}$$

$$D_{\tau=50}^{\text{pile cab}} = \begin{bmatrix} 0.9534 & 0.0237 & 0.0153 & 0.0075 & 0.0001 \\ 0 & 0.954 & 0.0231 & 0.0152 & 0.0077 \\ 0 & 0 & 0.9613 & 0.0233 & 0.0154 \\ 0 & 0 & 0 & 0.9767 & 0.0233 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{22}$$

$$D_{\tau=0}^{\text{floor}} = \begin{bmatrix} 0.9848 & 0.008 & 0.0049 & 0.0022 & 0.0001 \\ 0 & 0.9854 & 0.0074 & 0.0048 & 0.0024 \\ 0 & 0 & 0.9876 & 0.0075 & 0.0049 \\ 0 & 0 & 0 & 0.9926 & 0.0074 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{23}$$

$$D_{\tau=50}^{\text{floor}} = \begin{bmatrix} 0.9748 & 0.013 & 0.0081 & 0.004 & 0.0001 \\ 0 & 0.9754 & 0.0124 & 0.0081 & 0.0041 \\ 0 & 0 & 0.9793 & 0.0125 & 0.0082 \\ 0 & 0 & 0 & 0.9876 & 0.0124 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{24}$$

We interpolated the degradation matrices for $\tau = 1$ to $\tau = 49$, through the following equation:

$$P_{\tau=i} = P_{\tau=0} + \frac{i}{50-1}\left(P_{\tau=50} - P_{\tau=0}\right) \text{ for } i = 1, 2, ..., 49 \tag{25}$$

Our state space is discrete, with each component having five possible condition states. At each timestep t, the component either stays in the same state or transitions to a worse state, except for the last state, which is the *failed* state. The degradation rate also increases by one $\tau_{t+1} = \tau_t + 1$.

In our environment, we have modeled the three most occurring failure mechanisms found in the quay walls in Amsterdam. Each of these mechanisms yields a failure probability, determined by the number of components in a failed state corresponding to that particular mechanism.

With each failure mechanism, a set g of one or more components is involved; each failure mechanism can have multiple sets $g \in G$, and sets can share components. The probability of a failure mechanism occurring depends on the number of failing components in g.

The first failure mechanism involves the breaking of wooden poles. In this context, three poles can share a mutual relationship. The probability of failure for this mechanism is as follows:

$$F_{\text{poles}}(g) = \begin{cases} 0.12 & \#\{\text{failure} \in g\} = 1 \\ 0.20 & \#\{\text{failure} \in g\} = 2 \\ 0.60 & \#\{\text{failure} \in g\} = 3 \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

The second failure mechanism occurs when the pile cabs are in a failed state. The probability of this occurring is significantly higher if two adjacent pile cabs fail. The failure probability for this mechanism is:

$$F_{\text{pile cabs}}(g) = \begin{cases} 0.23 & \#\{\text{failure} \in g\} = 1 \\ 0.53 & \#\{\text{failure} \in g\} = 2 \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

The final mechanism pertains to the failure of the wooden floor. This particular mechanism is associated with a single component.

$$F_{\text{floor}}(g) = \begin{cases} 0.15 & \#\{\text{failure} \in g\} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

Each failure mechanic can be associated with multiple groups of components. We treat each failure mechanism as an independent event. Therefore, the final failure probability is calculated through:

$$\begin{aligned} p_f = 1 - &\prod_{g_p \in G_{\text{poles}}} (1 - F_{\text{poles}}(g_p)) \\ &\prod_{g_c \in G_{\text{pile cabs}}} (1 - F_{\text{pile cabs}}(g_c)) \\ &\prod_{g_f \in G_{\text{floor}}} (1 - F_{\text{floor}}(g_f)) \end{aligned} \tag{29}$$

In our experiments, we used the following component sets for the poles (G_{poles}), pile cabs ($G_{\text{pile cabs}}$), and floor (G_{floor}):

$$\begin{aligned} G_{\text{poles}} &= \{\{\text{comp}_1, \text{comp}_2, \text{comp}_3\}, \\ &\quad \{\text{comp}_4, \text{comp}_5, \text{comp}_6\}, \\ &\quad \{\text{comp}_7, \text{comp}_8, \text{comp}_9\}\} \\ G_{\text{pile cabs}} &= \{\{\text{comp}_{10}, \text{comp}_{11}\}, \\ &\quad \{\text{comp}_{11}, \text{comp}_{12}\}\} \\ G_{\text{floor}} &= \{\{\text{comp}_{13}\}\} \end{aligned} \tag{30}$$

The cost of repair and replacement is based on the percentage of full replacement of the quay walls. A repair is always 0.25 times the replacement cost. We use the information given by the city of Amsterdam to determine the following distribution of replacement cost of each type of component: Replacing a pole is $0.4 \times \frac{1}{9}$, replacing a pile cabs is $0.0375 \times \frac{1}{3}$, and replacing the floor is 0.1125. The summation of the replacement cost of all the components is not one because we do not plan maintenance on all the components of the quay wall, such as the brickwork. The cost of a global inspect action is 0.005. When an inspection action is taken, the observation will be the current state. However, when no inspection occurs, we still receive some information about the state. The city of Amsterdam is currently using satellite images that measure the movement of the quay wall. This data only reveals that the quay wall is in one of the three worst states but does not reveal much about the specific state. Therefore, if no inspection action is done, the observation matrix is:

$$O_{\text{no inspection}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.34 & 0.33 & 0.33 \\ 0 & 0 & 0.34 & 0.33 & 0.33 \\ 0 & 0 & 0.34 & 0.33 & 0.33 \end{bmatrix} \tag{31}$$

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:

$$s_0 = \{4, 4, 3, 4, 3, 3, 4, 3, 4, 3, 4, 3, 3\} \tag{32}$$

The initial belief of the state for each component is given in equation 33. We chose this distribution because, at the start, we did not know the state of the quay wall. Furthermore, we cannot determine well if the component is either heavily degraded (state 4) or failed (state 5).

$$\mathbf{b}_0 = \{0.25, 0.25, 0.25, 0.2, 0.05\} \tag{33}$$

Lastly, we set the constants for the utility function to $P = 4$, $C_{\max} = 2$, and $F_{\max} = 0.2$.

Hyperparameters The critic and actor network both used the `Tanh` activation function. The critic network has four layers (55, 100, 100, 121); the output layer is 121 due to $c = 11$. The actor network has one shared layer (55) and two layers for each component head (100, 4) and global head (100, 2). The learning rate for the critic is linear decayed from 1E-3 to 1E-4, and for the actor from 1E-4 to 1E-5. The V_{MIN} is set to -8 for the cost and -0.3 for the failure probability, and the V_{MAX} to 0 for both objectives. We update the weights after every 32 steps and set the clip of the grad norm at 10. We set the entropy coefficient to 0.1 and the value coefficient to 0.5. The discount factor is $\gamma = 0.975$, which is common for maintenance problems [2, 3]. We trained MODCMAC five times, each training run comprising 25 million steps.

Belief-State Based Policy In the experiments, we compared MODCMAC to a Belief-state-based (BSB) policy. This BSB policy calculates an action distribution for each component through the current belief $\mathbf{b}_t$. The action distribution is as follows with the order being {*nothing, repair, replace*}:

$$\{\mathbf{b}_{t,1} + \mathbf{b}_{t,2} * 0.25, \mathbf{b}_{t,2} * 0.75 + \mathbf{b}_{t,3}, \mathbf{b}_{t,4} + \mathbf{b}_{t,5}\} \tag{34}$$

We do the same for the global action, for which the distribution is as follows, with the order being {*nothing, inspect*}:

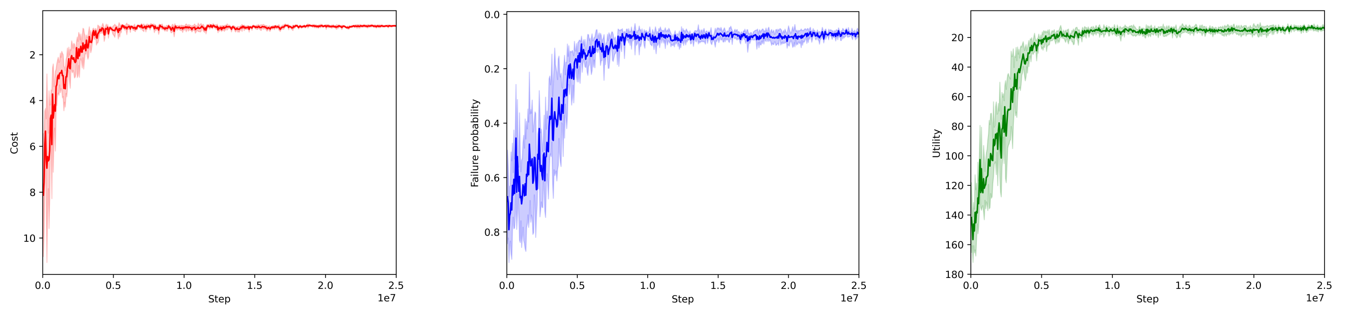
$$\{\mathbf{b}_{t,1} + \mathbf{b}_{t,2}, \mathbf{b}_{t,3} + \mathbf{b}_{t,4} + \mathbf{b}_{t,5}\} \tag{35}$$

The BSB policy then either takes the action with the highest probability in both distributions (equations 34 and 35) in the case of the deterministic version, **BSB-D**, or samples the action from these distributions in case of stochastic version, **BSB-S**.

Results The training results of MODCMAC show that it is able to learn a stable policy across the runs. We see that at the start, the utility score is quite large, hovering around 150 (Figure 2c), and displays a large standard deviation. We observe that the main contributor to this high standard deviation is the failure probability. In Figure 2b, we see that the standard deviation is significantly larger than the one found in cost (figure 2a). The most probable reason for this is the stochastic nature of our environment, in which the transition can differ significantly between episodes; therefore, the number of failed components can also differ in each episode.

In Table 1, we report the results of comparing MODCMAC to both BSB policies. The results show that MODCMAC achieves the best utility with 13.73±14.417, which is slightly better than BSB-S (14.694±17.166) and better than BSB-D (23.664±15.881). We also observed that the standard deviation for both the *utility* and *failure probability* is large for both BSB policies and MODCMAC. This can be primarily attributed to the stochastic nature of the environment and its impact on the *failure probability*. The significant variation in the *failure probability* consequently led to the large standard deviation observed in the *utility*.

If we look only at the best 25% utility scores, we see that BSB-D achieves the best utility (3.625) compared to MODCMAC (3.942). This indicates that BSB-D is likely the best policy in optimal situations in which no or almost no components go to the failed state but that it, on average, performs significantly worse. The reason for this is that BSB-D only selects a repairing or replacing action when it is sure that a component is in a bad state. This allows BSB-D to have the lowest cost (0.528±0.088). BSB-S has a significantly higher cost (1.029±0.103) but does have the lowest failure probability (0.064±0.116), especially compared to BSB-D (0.196±0.175).



(a) The average progression of the cost of MODCMAC over five runs. The shaded area is the standard deviation.

(b) The average progression of the failure probability score of MODCMAC over five runs. The shaded area is the standard deviation.

(c) The average progression of the utility score of MODCMAC over five runs. The shaded area is the standard deviation.

Figure 2: Learning curves for MODCMAC on our quay wall model.

While MODCMAC has neither the lowest cost (0.746±0.103) nor failure probability (0.071±0.112), it achieves the best trade-off according to our utility function, which resulted in the lowest utility score.

Table 1: The results of MODCMAC and the greedy method. We tested the Belief-state-based (BSB) policies (BSB-D and BSB-S) and each training run of MODCMAC for a 1000 episodes with different seeds. A lower score is desired for both the objectives and the utility. The best scores are highlighted in bold.

		Fail. Prob.	Cost	Util.
MODCMAC	Score	0.071±0.112	0.746±0.103	**13.73±14.417**
	25%	0	0.671	3.942
	50%	0	0.738	4.264
	75%	0.12	0.812	20.857
BSB-D	Score	0.196±0.175	**0.528±0.088**	23.664±15.881
	25%	0	0.46	3.627
	50%	0.15	0.532	19.502
	75%	0.322	0.588	38.617
BSB-S	Score	**0.064±0.116**	1.029±0.103	14.694±17.166
	25%	0	0.957	4.626
	50%	0	1.026	4.836
	75%	0.12	1.094	23.727

5 Related Work

To our knowledge, MODCMAC is the first deep MORL infrastructural maintenance planning method. However, there are multiple other single-objective RL approaches for infrastructural maintenance planning besides DCMAC [2]. Other research shows how a constrained POMDP can be applied in this setting, such that the risk and the budget are constrained [3]. This resembles our approach, in which we add penalties for going over certain values. However, the approach of Andriotis et al. [3] is on one side more sophisticated in that it will also work for step-wise budget constraints, meaning that it can only spend an allocated amount of budget at any timestep. This is currently not possible within MODCMAC. Nevertheless, Andriotis et al. [3] still need to define their failure probability or risk as a cost due to it being a single-objective RL method, whereas we can directly optimize over the failure probabilities. A similar approach is proposed in [11], where a DRL agent learns to group maintenance actions that are closer in proximity, resulting in reduced repair and replacement costs of neighboring assets. In our approach, this could be rewritten as a multi-objective problem, in which one of the objectives is to group maintenance actions. Other recent approaches utilized Hierarchical Reinforcement Learning (HRL) for maintenance planning [1, 8] to address the scalability issue with the action space. Besides the HRL contribution, Hamida et al. [8] utilize a continuous state space instead of a discrete state space for the deterioration model. We see the usage of a continuous state space also in other research [13, 15].

While MODCMAC might be the first deep MORL approach for a multi-objective maintenance planning problem, it is not the first multi-objective approach to it. A common method for maintenance planning is to use standard optimization techniques such as genetic algorithms [5, 6, 7]. However, our proposed method will likely perform better with known non-linear utility functions.

6 Conclusion and Future Works

In this paper, we introduced MODCMAC an MORL approach to optimally plan maintenance of multi-component structure. We showed how our method is able to learn a non-linear utility function, in an ESR setting, by attaining the most optimal utility score, when compared to the Belief-state-based policies. In future research, we plan to test our method with other and more complex maintenance environments. We want to do this by adding multiple assets, while still planning the maintenance at the component level for each asset. Moreover, we want to increase the complexity of the environment by adding complex interactions among assets and increasing the number of possible actions to include different lifespan extension measures, and diverse interventions such as limiting heavy traffic, reducing speed, etc. By adding these actions, we also want to include more objectives. For example, with lifespan extension methods, we could limit emissions and add this as an objective that needs to be minimized while limiting traffic could be added as an availability objective that needs to be maximized. However, MODCMAC's critic network output grows exponentially to a number of objectives. This exponential increase of the critic output would increase training time significantly. Therefore, we need to investigate other approaches to accommodate a larger number of objectives [10].

Acknowledgements

This research is partially supported by funding from the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” program, and the Dutch Research Council

(NWO) and the municipality of Amsterdam under the Urbiquay program of the STABILITY and LiveQuay projects. Personal thanks to L. Smalbil, who helped review the paper.

References

[1] Charalampos P. Andriotis. and Ziead Metwally., '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, (2023).

[2] C.P. Andriotis and K.G. Papakonstantinou, 'Managing engineering systems with large state and action spaces through deep reinforcement learning', *Reliability Engineering & System Safety*, **191**, 106483, (2019).

[3] C.P. Andriotis and K.G. Papakonstantinou, 'Deep reinforcement learning driven inspection and maintenance planning under incomplete information and constraints', *Reliability Engineering & System Safety*, **212**, 107551, (2021).

[4] Robert Borgovini, Stephen Pemberton, and Michael Rossi, 'Failure mode, effects and criticality analysis (fmeca)', *Reliability Analysis Center*, (1993).

[5] Zaharah Allah Bukhsh, Irina Stipanovic, and Andre G. Doree, 'Multi-year maintenance planning framework using multi-attribute utility theory and genetic algorithms', *European Transport Research Review*, **12**(1), (jan 2020).

[6] You Dong, Dan M. Frangopol, and Duygu Saydam, 'Pre-earthquake multi-objective probabilistic retrofit optimization of bridge networks based on sustainability', *Journal of Bridge Engineering*, **19**(6), 04014018, (2014).

[7] Jinchao Guan, Xu Yang, Lingyun You, Ling Ding, and Xiaoyun Cheng, 'Multi-objective optimization for sustainable road network maintenance under traffic equilibrium: Incorporating costs and environmental impacts', *Journal of Cleaner Production*, **334**, 130103, (2022).

[8] Zachary Hamida and James-A. Goulet, 'Hierarchical reinforcement learning for transportation infrastructure maintenance planning', *Reliability Engineering System Safety*, **235**, 109214, (2023).

[9] Conor F. Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai A. Irissappane, Patrick Mannion, Ann Nowé, Gabriel Ramos, Marcello Restelli, Peter Vamplew, and Diederik M. Roijers, 'A practical guide to multi-objective reinforcement learning and planning', *Autonomous Agents and Multi-Agent Systems*, **36**(1), 26, (Apr 2022).

[10] Conor F. Hayes, Timothy Verstraeten, Diederik M. Roijers, Enda Howley, and Patrick Mannion. Multi-objective coordination graphs for the expected scalarised returns with generative flow models, 2022.

[11] David Kerkkamp., Zaharah A. Bukhsh., Yingqian Zhang., and Nils Jansen., 'Grouping of maintenance actions with deep reinforcement learning and graph convolutional networks', in *Proceedings of the 14th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART*, pp. 574–585. INSTICC, SciTePress, (2022).

[12] M. Korff, M. Hemel, and D.J. Peters, 'Collapse of the grimburgwal, a historic quay in amsterdam, the netherlands', *Proceedings of the Institution of Civil Engineers - Forensic Engineering (online)*, **175 (4)**, 96–105, (2022).

[13] Christos Lathourakis, Charalampos P. Andriotis, and Alice Cicirello, '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, (2023).

[14] Mathieu Reymond, Conor F. Hayes, Denis Steckelmacher, Diederik M. Roijers, and Ann Nowé, 'Actor-critic multi-objective reinforcement learning for non-linear utility functions', *Autonomous Agents and Multi-Agent Systems*, **37**(2), 23, (2023).

[15] Erotokritos Skordilis and Ramin Moghaddass, 'A deep reinforcement learning approach for real-time sensor-driven decision making and predictive analytics', *Computers Industrial Engineering*, **147**, 106600, (2020).

[16] Harold Soh and Yiannis Demiris, 'Evolving policies for multi-reward partially observable markov decision processes (mr-pomdps)', in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 713–720, (2011).

[17] Matthijs T. J. Spaan, *Partially Observable Markov Decision Processes*, 387–414, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[18] C Ch White and KIM KW, 'Solution procedures for vector criterion markov decision processes', *Large Scale Systems*, **1**, 129–140, (1980).

[19] Tiago Zonta, Cristiano André da Costa, Rodrigo da Rosa Righi, Miromar José de Lima, Eduardo Silveira da Trindade, and Guann Pyng Li, 'Predictive maintenance in the industry 4.0: A systematic literature review', *Computers Industrial Engineering*, **150**, 106889, (2020).