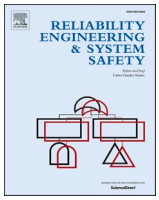
Reliability Engineering and System Safety 239 (2023) 109512

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress



Risk-informed operation and maintenance of complex lifeline systems using parallelized multi-agent deep Q-network

Dongkyu Lee, Junho Song *

Department of Civil and Environmental Engineering, Seoul National University, Seoul, South Korea

ARTICLE INFO

Keywords:
Deep reinforcement learning
Lifeline systems
Life-cycle cost
Markov decision process
Operation & maintenance
Parallel processing

ABSTRACT

Lifeline systems such as transportation and water distribution networks may deteriorate with age, raising the risk of system failure or degradation. Thus, system-level sequential decision-making is essential to address the problem cost-effectively while minimizing the potential loss. Researchers have proposed to assess the risk of lifeline systems using Markov decision processes (MDPs) to identify a risk-informed operation and maintenance (O&M) policy. In complex systems with many components, however, it is potentially intractable to find MDP solutions because the numbers of states and action spaces increase exponentially. This paper proposes a multi-agent deep reinforcement learning framework, termed parallelized multi-agent deep Q-network (PM-DQN), to overcome the curse of dimensionality. The proposed method takes a divide-and-conquer strategy, in which multiple subsystems are identified by community detection, and each agent learns to achieve the O&M policy of the corresponding subsystem. The agents establish policies to minimize the decentralized cost of the cluster unit, including the factorized cost. Such learning processes occur simultaneously in several parallel units, and the trained policies are periodically synchronized with the best ones, thereby improving the master policy. Numerical examples demonstrate that the proposed method outperforms baseline policies, including conventional maintenance schemes and the subsystem-level optimal policy.

1. Introduction

Components in civil infrastructure systems, such as bridges, pipes, and electric wires, deteriorate over the life cycle of the systems due to corrosion of materials and other environmental factors. Component failures caused by the degradation may trigger a system-level failure event, e.g., power outage, service disruption, loss of network connectivity, or degenerate the systems' the quality of service (QoS) like usability and serviceability [1]. Since civil infrastructure systems, especially lifeline systems including power, water, or gas transmission systems, have a great impact on modern societies, not only can each failure cause huge socio-economic losses, but system-level malfunctions multiply the damage exponentially. To quantify the risk uncertainty of these system elements, numerous studies on risk modeling based on stochastic processes have been conducted [2,3]. Based on these modeled risk, decision-makers should establish apposite risk-informed operation and maintenance (O&M) policies considering the inevitable changes in system-level QoS and the consequences of system failure [2,4].

Various O&M strategies have been developed for structures in civil lifeline systems. For example, time-based maintenance (TBM) undertakes maintenance based on the predetermined repair time intervals, i.e., without monitoring or criteria evaluation [5,6]. On the other hand, condition-based maintenance (CBM) aims to perform preventive maintenance based on the information acquired or interpreted in various forms [6,7,8,9]. Although it is straightforward to scale and apply these O&M schemes to multiple components, the introduction of exogenous variables such as the cost of system failure events hampers their direct extensions to system-level decision-making processes. This is because component-level optimality does not ensure the optimal decisions at the system level [10]. Therefore, stakeholders and decision-makers of civil lifeline systems should be able to establish an O&M policy based on the system-level performance rather than on the conditions of individual components during the life cycle.

For O&M policy planning for a system consisting of components subject to uncertain failures, classical theories of system reliability [11,12,13] often introduce a Markovian model to handle the characteristics of the components and the system. If the same system control policy is applied over the life cycle, stationary conditions are achieved after a

* Corresponding author.
E-mail address: junhosong@snu.ac.kr (J. Song).

https://doi.org/10.1016/j.ress.2023.109512
Received 20 July 2022; Received in revised form 5 May 2023; Accepted 16 July 2023
Available online 17 July 2023

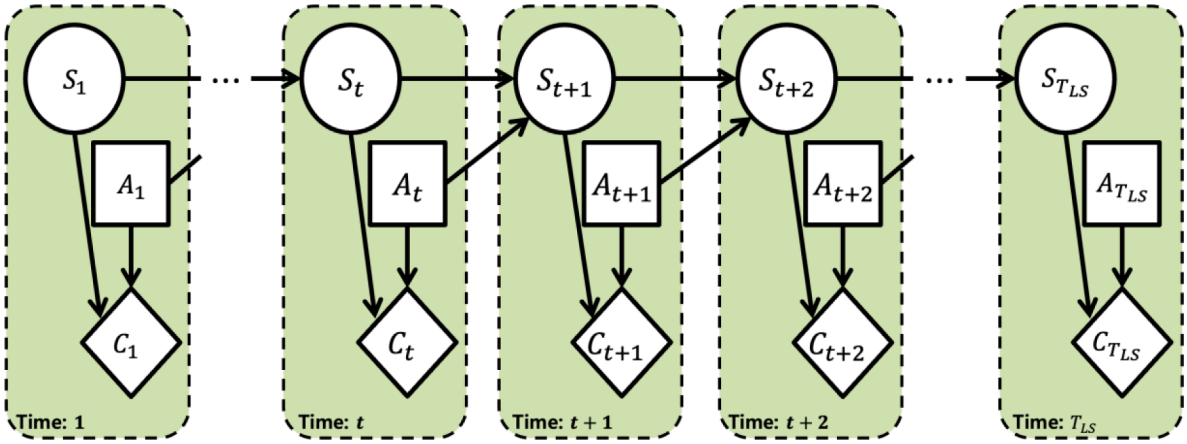


Fig. 1. Graph of a Markov decision process (MDP) model.

Table 1
Flow capacity of each component for each damage state.

Damage state	AGAN	Slight	Moderate	Extensive	Collapse
Flow Capacity	1.00	0.95	0.50	0.25	0

warm-up period. Thus, the system availability can be expressed as a closed-form expression under the assumption that components have constant mean failure rates [13]. However, such an O&M policy based on the steady-state assumption is not flexible enough to cope with many plausible exceptions in practice, e.g., an old road facing its service life.

In efforts to obtain optimal O&M decisions without special assumptions such as steady-state conditions in a finite time horizon, a Markov decision process (MDP) has been considered as a preferred framework to formulate the stochastic processes of structural deterioration and the risk-informed O&M optimization. Because of the capability to quantify the expected cost or value of various maintenance actions in each state through dynamic programming, MDPs have been applied to establish the O&M policies for multi-state systems [14,15] as well as component-level policies [16] for civil infrastructures. Although MDPs have a fundamental limit in direct applications to large systems due to the curse of dimensionality, i.e., the state and action spaces increase exponentially with the number of the components, researchers have proposed relaxing constraints or seeking approximate solutions [17,18].

In particular, to overcome the challenges, recent studies have applied many deep reinforcement learning (DRL)-based algorithms [19,20,21,22] to maintenance optimization problems in civil infrastructure systems [23,24,25]. More specifically, Yang et al. [26] utilized a deep Q-learning (DQN) framework for the optimal CBM planning of a multi-state system, and the framework has been further refined with the introduction of double DQN (DDQN) [27,28,29]. Moreover, multi-agent reinforcement learning (MARL) [30,31,32,33], in which multiple agents learn policies to achieve the minimum costs using DRL, has been considered to improve the scalability of DRL applications for optimizing the maintenance of civil systems. For example, Andriotis and Papakonstantinou [10,34] have developed centralized or decentralized MARL-based algorithms, respectively, in terms of sharing or not sharing parameters related to each agent's policy for large civil systems. Zhou et al. [35] also proposed MARL-based hierarchical algorithms to optimize the O&M decisions of large-scale multi-component systems. To enhance the performance of MARL, Nguyen et al. [36] introduced a value decomposition network (VDN) algorithm [32,37], decomposing the centralized Q-value into the sum of decentralized Q-values defined for individual agents, to find the best O&M decisions. However, although these MARL algorithms successfully reduce the computational complexity, model training or hyperparameter tuning still requires considerable time even in the relaxed environment. In the numerical examples where the existing MARL methods are applied, the topologies are quite monotonous, such as series- or parallel-systems, and the number of components is also limited.

In this paper, we propose a parallelized multi-agent deep Q-network (PM-DQN), a divide-and-conquer algorithm that utilizes clustering-based multi-scale approaches [38,39], for risk-informed decision optimization of multi-state flow systems by combining the system simplification with MARL based on parallel processing. In PM-DQN, the agents observe the states of multiple subsystems identified by clustering, and learn decentralized policies to minimize the factorized cost for subsystems. The algorithm guides the decentralized policies chosen by the agents toward the system-level optimality by repeating the learning process through parallel processing and tuning a key hyperparameter based on the results with less computational cost. As a result, the PM-DQN can efficiently manage large systems that are out of control in existing approaches.

This paper is organized as follows. Section 2 provides a description of the problem setting and the model assumptions for numerical examples, discusses the system-level sequential decision-making optimization problem, and introduces the DDQN and MARL frameworks. Section 3 proposes the PM-DQN algorithm based on the configuration of subsystems based on community detection and the cost factorization. The efficiency and superior performance of the proposed algorithm are demonstrated by numerical examples in Section 4, followed by a summary and concluding remarks in Section 5.

2. Background information

2.1. Problem statement

A finite-time Markov decision process (MDP) shown in Fig. 1 is defined as a five-tuple $(\boldsymbol{S}, \boldsymbol{A}, T, C, \gamma)$, the elements of which are described below.

2.1.1. Descriptions of system state

In this study, we consider a multi-state flow system with n components. It is assumed that the initial state of each component is intact, i.e., as-good-as-new (AGAN) state. The degradation level of components is identified as one of the following discretized damage states through annual observations: *AGAN, slight damage, moderate damage, extensive damage,* and *collapse*. We assume that the system state is observed without errors. The system degradation state $\boldsymbol{S}_t = [s_t^i]_{i\in[1,n]}$ is given as a vector of component states, where s_t^i is the state of component $i \in [1, n]$ at time t.

As a performance indicator, the maximum flow capacity between

two predetermined terminals is used to quantify the QoS of the system. It is assumed that the flow capacity of all components depends on each state as shown in Table 1. Once the flow capacities of all components are determined, i.e., the system state $\boldsymbol{S}_t$ is observed, the loss of the system QoS, $L_{sys}(\boldsymbol{S}_t)$, can be calculated as the difference between the current maximum flow capacity and the maximum flow capacity under the initial system state through the max-flow algorithm [1,40,41]. Depending on the topology of the system, each component's flow capacity loss due to the deterioration has a different effect on the QoS of the system.

2.1.2. Descriptions of maintenance and degradation process

Based on the system state information, the agent chooses one of the two options for O&M action a_t^i for the component $i \in [1, n]$ at time t, i.e., 'Do Nothing (*DN*)' and 'Repair it to AGAN state (*R*).' System-level action $\boldsymbol{A}_t = [a_t^i]_{i\in[1,n]}$ is defined as a vector of the actions taken for individual components. Although the size of the action space seems small at the component level, the number of system-level actions that a decision-maker considers each year is 2^n, which increases exponentially with the number of components.

We assume that there is no limit to the number of components that can be repaired per step. When the decision-maker selects *R* for one component, the state becomes the AGAN state deterministically, otherwise it remains the same or deteriorates with a certain probability called state-transition probability T. The stochastic degradation process of individual components is modeled as an independent discrete-time Markov process. That is, the state-transition probability $T(s_{t+1}^i | s_t^i, a_t^i)$ for component $i \in [1, n]$ depends only on the current state s_t^i and action a_t^i, due to its Markov properties. While the state-transition probabilities for *R* are the same for all components, the state-transition probabilities for *DN* are modeled in two ways depending on the component types, each shown as 5×5 matrices in the Appendix A. Once the system-level action $\boldsymbol{A}_t$ for all components is determined in system state $\boldsymbol{S}_t$, we can define the system-level state-transition probability $T(\boldsymbol{S}_{t+1}|\boldsymbol{S}_t, \boldsymbol{A}_t)$ based on the state-transition probabilities $T(s_{t+1}^i | s_t^i, a_t^i)$ for all components.

2.1.3. Descriptions of operation and maintenance costs

In the O&M process of civil lifeline systems, various variables (e.g., maintenance costs, system QoS, component malfunction) must be considered simultaneously at each time step. Because of trade-offs between life-cycle costs and the risk of systems, there is often no single solution that optimizes all these variables simultaneously. It is necessary to find an appropriate compromise among the conflicting goals. In multi-objective optimization, one can explore Pareto-optimal solutions that cannot improve an objective function without degrading other objective functions [42]. However, to make a decision among Pareto-optimal solutions, the subjective preference of decision-makers must intervene inevitably.

One can handle this issue by transforming multiple objective functions into a single objective function through scalarization, i.e., the weighted sum formulation. In this approach, the optimal solution strongly depends on cost functions (or utility functions) used in the scalarization process. In this paper, the objective function c_{tot}, which is the total cost at time t, is defined as the sum of the multiple objective functions scaled to cost as follows:

$$c_{tot}(\boldsymbol{S}_t, \boldsymbol{A}_t) = \sum_i \left[c_{M,i}(a_t^i) + c_{S,i}(s_t^i)\right] + c_{SD}(\boldsymbol{S}_t), \tag{1}$$

where $c_{M,i}$ is the maintenance cost for the component i; $c_{S,i}$ is the cost due to the shutdown of the component i; and c_{SD} is the system damage cost caused by the loss of system QoS. In Eq. (1), $c_{M,i}$ conflicts with the other operation costs; $c_{S,i}$ and c_{SD}, which depend on the state of components and system, decrease with a conservative O&M decision, while $c_{M,i}$ incurred by repairs increases. Therefore, a suitable compromise between these costs is desirable.

In numerical examples in Section 4, it is assumed that the maintenance cost $c_{M,i}$ of 1.0 is imposed when a_t^i is *R* regardless of s_t^i, and the shutdown cost $c_{S,i}$ of 1.0 is charged when component i reached the collapse state. Loss of the system QoS, $L_{sys}(\boldsymbol{S}_t)$, penalizes the system damage cost c_{SD} by a factor of f_{SD} as

$$c_{SD}(\boldsymbol{S}_t) = f_{SD} \cdot L_{sys}(\boldsymbol{S}_t) = f_{SD} \cdot \left[MF_{sys}(\boldsymbol{S}_0) - MF_{sys}(\boldsymbol{S}_t)\right], \tag{2}$$

where $MF_{sys}(\boldsymbol{S}_t)$ is a function representing the maximum flow capacity between predetermined two terminals under system state $\boldsymbol{S}_t$; and $\boldsymbol{S}_0$ is the initial system state vector, i.e., every component is in AGAN state.

If the c_{SD} term in Eq. (1) is neglected, i.e., the objective function is given as a linear function of the component-level costs, the system-level optimization problem is decomposed into multiple component-level optimization problems. The optimal solutions to the subproblems provide the system-level optimum. In most cases, however, c_{SD} given as a function of component state combinations is of great interest. Therefore, the objective function c_{tot} cannot be decomposed into component-level decentralized costs. In other words, the number of combinable policies to explore for achieving a system-level optimal solution grows exponentially with the system size.

2.2. System-level sequential maintenance optimization

In maintenance optimization problems for civil lifeline system, the agent's goal is to determine a policy $\pi(\boldsymbol{A}_t|\boldsymbol{S}_t)$ as a mapping from the state $\boldsymbol{S}_t$ to the action $\boldsymbol{A}_t$ to minimize the Q-value, defined as the expected total discounted costs to the system's lifespan T_{LS} as

$$Q_\pi(\boldsymbol{S}_t, \boldsymbol{A}_t) = E_\pi\left[\sum_{k=t}^{T_{LS}} \gamma^{k-t} c_{tot}(\boldsymbol{S}_k, \boldsymbol{A}_k) | \boldsymbol{S}_t, \boldsymbol{A}_t\right], \tag{3}$$

where $\gamma \in [0, 1]$ is the discount factor to convert future costs into present values. Just as the component-level optimality does not guarantee the system-level optimality, a greedy optimal decision at every time step cannot assure optimality over the long run in a finite-time horizon environment. This applies to sequential maintenance optimization problems for civil lifeline system O&M throughout the life cycle. An O&M policy π is defined to be better than or equal to another O&M policy π' if and only if its expected cost is smaller than or equal to that of π' for all states [43]. That is, the optimal O&M policy π^* leads the Q-value Q_{π^*} to its minimum.

In value iteration [43,44], considering this point, the Q-value $Q_\pi(\boldsymbol{S}_t, \boldsymbol{A}_t)$ is updated by continuously choosing the action with the lowest Q-value as

$$\pi^*(A_t|S_t) = \begin{cases} 1, \text{if } \boldsymbol{A}_t = \underset{\boldsymbol{A}'}{\operatorname{argmin}} Q_{\pi^*}(\boldsymbol{S}_t, \boldsymbol{A}') \\ 0, \text{otherwise.} \end{cases} \tag{4}$$

After initializing all the Q-value estimates to zero, the values are updated iteratively according to the following Bellman equation:

$$Q_{k+1}(\boldsymbol{S}_t, \boldsymbol{A}_t) = c_{tot}(\boldsymbol{S}_t, \boldsymbol{A}_t) + \gamma \sum_{\boldsymbol{S}'} \left[T(\boldsymbol{S}'|\boldsymbol{S}_t, \boldsymbol{A}_t) \min_{\boldsymbol{A}'} Q_k(\boldsymbol{S}, \boldsymbol{A}')\right], \tag{5}$$

where $Q_k(\boldsymbol{S}_t, \boldsymbol{A}_t)$ is the Q-value on state $\boldsymbol{S}_t$ and action $\boldsymbol{A}_t$ estimated at the k^{th} iteration. The value iteration algorithm works well in simple environments but does not apply to environments with large state and action spaces. In particular, evaluating the Q-value of every combinable state-action pair in complex environments is nearly impossible.

2.3. Double Deep Q-Network with prioritized experience replay

To overcome the computational limitation, Watkins and Dayan [22] proposed to apply a model-free reinforcement learning algorithm called

Q-learning. In the Q-learning algorithm, instead of Eq. (5), the following equation is used to update Q-values iteratively:

$$Q_{k+1}(\boldsymbol{S}_t,\boldsymbol{A}_t) = Q_k(\boldsymbol{S}_t,\boldsymbol{A}_t) + \alpha\cdot[y_t - Q_k(\boldsymbol{S}_t,\boldsymbol{A}_t)], \tag{6}$$

where $\alpha \in (0,1)$ is the learning rate; and $y_t = c_{tot}(\boldsymbol{S}_t,\boldsymbol{A}_t) + \gamma \min_{\boldsymbol{A}'} Q_k(\boldsymbol{S}_{t+1}, \boldsymbol{A}')$ is the target value.

The ε-greedy algorithm is often implemented to expedite the exploration and exploitation process for Q-learning [43]. In detail, after initializing Q-value to zero, an agent selects a random action (i.e., exploration) with probability ε, or the action minimizing Q-value (i.e., exploitation) with probability $1-\varepsilon$. It is common to start with the value of ε close to 1 and gradually decrease it as information about the environment accumulates.

In a powerful deep reinforcement learning (DRL) framework developed by Mnih et al. [19,20], a parameterized deep Q-network (DQN) is introduced to approximate the Q-value. Each layer in the DQN outputs a linear combination of inputs and parameters. The framework introduces activation functions between layers, such as rectified linear unit (ReLU), exponential linear unit (ELU), and scaled ELU (SELU), to describe the complex relationships beyond linearity. However, DQN tends to underestimate the Q-values in stochastic environments due to the 'min' operator in calculating the target value y_t. Double DQN (DDQN) [27] performs the action selection and Q-value evaluation separately using two different DQNs (i.e., online and target networks), thereby avoiding the underestimation of the Q-values and stabilizing the learning process. The target value y_t is rewritten as

$$y_t = c_{tot}(\boldsymbol{S}_t,\boldsymbol{A}_t) + \gamma Q_t\left(\boldsymbol{S}_{t+1}, \operatorname*{argmin}_{\boldsymbol{A}'} Q_o(\boldsymbol{S}_{t+1},\boldsymbol{A}';\theta_k);\theta_k^-\right), \tag{7}$$

where Q_o is the Q-value estimated using the online network with parameters θ_k; and Q_t is the Q-value estimated using the target network with parameters θ_k^-, respectively. The parameters θ_k^- are periodically updated to the parameters θ_k every C steps.

In addition, one can introduce the experience replay [45], in which agents store the experiences as tuples $e_t = (s_t, a_t, r_t, s_{t+1})$ in a replay buffer D, and update the online network parameters θ_k based on a batch of uniformly sampled tuples from the replay buffer. The uniformly drawn samples significantly reduce the variance of updates owing to the weakened correlation, thereby suppressing the oscillation or divergence of the parameters θ_{k+1} during the training process. Combining these techniques, DDQN calculates the loss function $L(\theta_k)$ and updates the online network parameters θ_{k+1} to minimize $L(\theta_k)$ by the gradient descent as follows:

$$L(\theta_k) = E_{e_t\sim U(D)}\left[(y_t - Q_o(\boldsymbol{S}_t,\boldsymbol{A}_t;\theta_k))^2\right], \tag{8}$$

$$\begin{aligned}\theta_{k+1} &= \theta_k - \frac{1}{2}\alpha\nabla_{\theta_k}L(\theta_k)\\ &= \theta_k + \alpha E_{e_t\sim U(D)}\left[(y_t - Q_o(\boldsymbol{S}_t,\boldsymbol{A}_t;\theta_k))\nabla_{\theta_k}Q_o(\boldsymbol{S}_t,\boldsymbol{A}_t;\theta_k)\right],\end{aligned} \tag{9}$$

where $U(D)$ represents a uniform distribution over the replay buffer D; and e_t is a batch of the tuples sampled from $U(D)$.

Agents can gauge the importance of each experience by the temporal difference error (TD-error), i.e., the difference between the expected Q-values before and after the experience. In prioritized experience replay (PER) [46], the alternative sampling density $h(\cdot)$ of a batch of experience tuples $e_t = (s_t, a_t, r_t, s_{t+1})$, which was uniformly distributed, is given as:

$$h(e_t;\alpha) = \frac{|\delta_t|^\alpha}{\sum_t |\delta_t|^\alpha}, \tag{10}$$

where $\delta_t = y_t - Q_o(\boldsymbol{S}_t,\boldsymbol{A}_t;\theta_k)$ is the TD-error; and $\alpha \geq 0$ is an importance sampling hyperparameter, with $\alpha = 0$ corresponding to the existing experience replay. The loss function $L(\theta_k)$ in Eq. (8) can be calculated more efficiently using importance sampling as follows:

$$L(\theta_k) = E_{e_t\sim U(D)}\left[\delta_t^2\right] = E_{e_t\sim h}\left[\delta_t^2\cdot\frac{f_U(e_t)}{h(e_t;\alpha)}\right] = E_{e_t\sim h}\left[\delta_t^2\cdot W(e_t;\alpha)\right], \tag{11}$$

where $f_U(\cdot)$ denotes the probability mass function of the discrete uniform distribution; and $W(e_t;\alpha) = f_U(e_t)/h(e_t;\alpha)$ is the likelihood ratio to compensate for the bias caused by introducing the alternative sampling density $h(e_t;\alpha)$. The initial learning process is highly non-stationary because the online and target networks have not been trained enough. Therefore, even if the estimate is slightly biased, stabilization should be the top priority at the beginning phase of learning, and importance sampling correction may be considered later. To this end, $W(e_t;\alpha)$ in Eq. (11) is replaced by $\widetilde{W}(e_t;\alpha,\beta)$, defined as

$$\widetilde{W}(e_t;\alpha,\beta) = \left(\frac{f_U(e_t)}{h(e_t;\alpha)}\right)^\beta = (N\cdot h(e_t;\alpha))^{-\beta}, \tag{12}$$

where N is the number of experiences in the replay buffer D; and $\beta \in [0,1]$ is a hyperparameter controlling how much to compensate for the bias. As β is gradually annealed toward 1.0, starting from a low value (e.g., typically around 0.4 to 0.6), the bias is completely compensated. When combined, DDQN and PER exert a substantial synergistic effect, thereby neutralizing a significant portion of the existing limitations of Q-learning algorithms.

2.4. Factorization in multi-agent reinforcement learning

In addition to increasing the efficiency of DRL like DDQN with PER, dividing the original problem into multiple subproblems can be a more effective solution to overcome the curse of dimensionality caused by the exponentially increasing number of states and actions. A divide-and-conquer strategy called a multi-agent reinforcement learning (MARL) deploys multiple agents in each segmented state and action spaces, and the agents learn policies to achieve the minimum costs independently or through cooperation.

If there are no cost functions that agents jointly contribute, a given environment can be simplified by decomposing it into several smaller independent environments [30,47]. In other words, the optimal decision set in sub-problems yields the same solution as the optimal global decision-making. Then, the size of the action space is shrunk from $\prod_{j=1}^{m}|A^j|$ to $\sum_{j=1}^{m}|A^j|$, where m is the number of agents, and $|A^j|$ is redefined as the number of available actions for the j^{th} agent. As a result, one can handle environments with large spaces of states and actions based on the large scalability.

However, in most of complex environments, cost functions take a combination of various states and actions as input, e.g., Eq. (1). This leads to the multi-agent credit allocation problem [33], in which the contributions of individual agents to the cost function must be accurately inferred. To this end, the problem can be segmented into multiple independent environments by factorizing the centralized cost into the sum of decentralized costs through neural networks called value decomposition networks (VDNs) [32] or user-defined functions. No matter how sophisticated the original cost is segmented, accuracy loss is inevitable during the factorization process, thereby resulting in a trade-off between accuracy and scalability.

3. Proposed algorithm: parallelized multi-agent deep Q-network (PM-DQN)

The MARL method combined with DDQN and PER improves the scalability with minimal loss of accuracy, allowing the MDP framework to be applied to complex environments. However, as the number of components in the system increases, the accuracy loss of the MARL method gradually accumulates, eventually leading to a failure in optimal policy search. To alleviate the scalability issue, in this section,

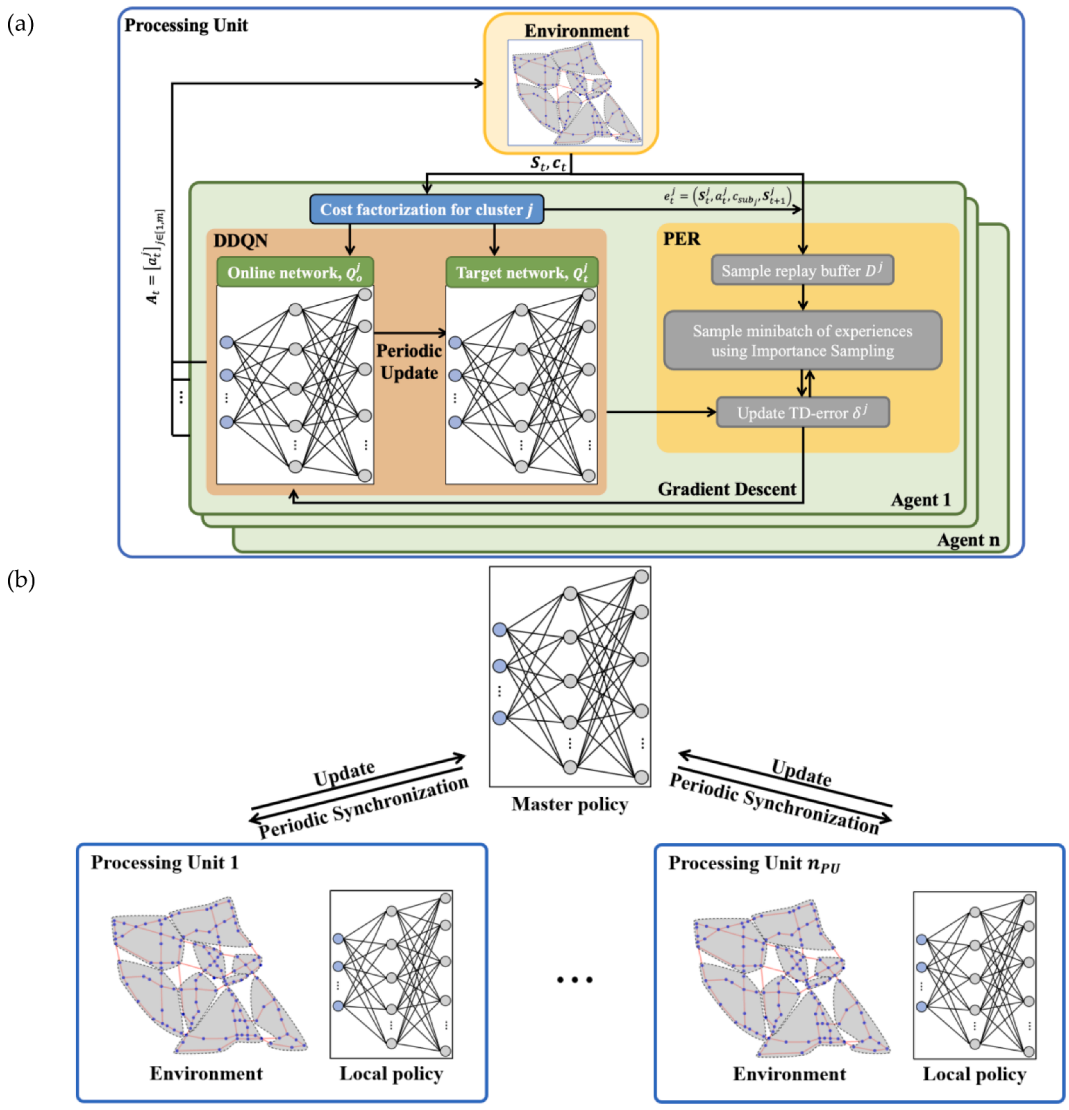


Fig. 2. (a) Learning process using DDQN with PER in a processing unit, and (b) overall procedure of Parallelized Multi-agent Deep Q-Network (PM-DQN) algorithm.

we propose a novel MARL algorithm termed "Parallelized Multi-agent Deep Q-Network" (PM-DQN), which assigns multiple agents to subsystems based on community detection, and makes the agents explore decentralized cost-minimizing policies under various hyperparameter values in multiple processing units. The proposed algorithm consists of three steps differentiated from others as follows: (1) system simplification based on network clustering for effective agent allocation, (2) factorization of centralized costs based on a predefined function, and (3) the hyperparameter tuning with parallel processing. Fig. 2(a) and (b) respectively illustrate the proposed algorithm's microscopic and macroscopic structures of operations.

3.1. Step 1: identification of subsystems based on community detection

In optimizing MARL-based O&M strategies in large-scale systems, it is necessary to find an appropriate balance between minimizing accuracy loss and efficient reduction of the state and action spaces. To this end, we propose a decomposition of the system into a few subsystems, i.e., communities of components. By grouping deeply related components in terms of functionality or densely connected components into one subsystem, the existing system can be simplified into another system of subsystems. This can also lower computational complexity that may scale exponentially with the number of components, while minimizing loss of information. Various algorithms such as spectral clustering algorithms [38,48] and Markov clustering algorithm [49,50] have been proposed for community detection. In this paper, the target system is represented by m subsystems instead of the existing n components based on the Girvan-Newman algorithm [39,51].

The Girvan-Newman algorithm sequentially removes edges with the highest edge-betweenness, which means the probability that the shortest paths between all node pairs in the graph go through the edge [52]. As edges with high edge-betweenness are removed, the entire graph representing the system is divided into several isolated clusters, and the process of edge removal ends when all edges are removed. In Girvan-Newman algorithm, the modularity M is introduced to stop the process at an appropriate time [51]. The modularity represents the difference between the actual number of intra-cluster edges in the existing graph and the expected number of intra-cluster edges when reconstructing the graph while preserving the degree of each node, and is defined as

$$M = \sum_{c\in[1,n_c]} \left[\frac{L_c}{n_e} - \left(\frac{k_c}{2n_e} \right)^2 \right], \tag{13}$$

where n_c is the number of isolated clusters; n_e is the number of edges in the graph; L_c is the number of intra-cluster edges in cluster c; and k_c is

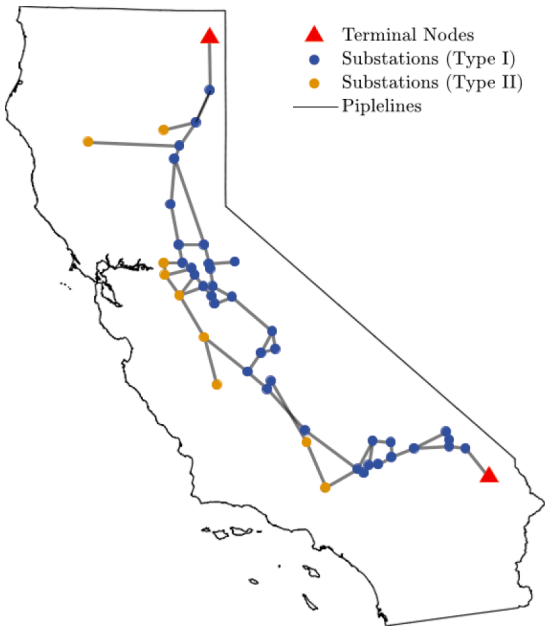


Fig. 3. California gas distribution system with 48 multi-state gas substations.

the sum of degrees of the nodes in cluster c. Modularity is updated whenever an edge is removed, and when it is maximized, one can identify subsystems that are separated from each other.

Once m subsystems in the target system are identified using the Girvan-Newman algorithm, we can assign an agent to learn the O&M strategy of each subsystem. However, there is a problem in utilizing the community detection result for PM-DQN; since all agents must be trained for optimal decision-making for the system-level optimal policy, the convergence of PM-DQN is governed by the slowest learning rate among agents (generally the agent with the largest state and action spaces). If components are concentrated in specific clusters, the learning time of the agents increases exponentially, thereby leading to a decrease in learning efficiency of PM-DQN. Therefore, a procedure for adjusting the subsystems to an even size is necessary. To this end, after initial grouping based on the Girvan-Newman algorithm, some components in the largest subsystem are reallocated to other neighboring subsystems. In this process, the number of edges connecting subsystems should be minimized to prevent loss of calculation accuracy due to system simplification.

Fig. 3 shows a simplified California gas distribution system (modified from [53]) with 48 gas substations of two types and 60 bidirectional pipelines. It is assumed that the pipelines are intact during the lifetime of the system, and only the deterioration of the gas substations, whose states can be periodically identified, is considered. The system can be described by a graph model with 48 nodes and 60 edges. Table 2 shows how modularity and the number of detected clusters change with edge removal by the Girvan-Newman algorithm. As the edges are removed continuously, the number of clusters increases, but the modularity peaks at 8 clusters. However, the detected clusters consist of five to seven components as shown in Fig. 4(a), resulting in differences in the action space of the agents. Since the learning rate of the agent managing the largest subsystem is lower than that of the other agents, the benefits of parallel processing in simultaneous training of all agents can be underutilized. To compensate for this limitation, one component is reallocated from the largest subsystem to the neighboring smallest subsystem 3 as shown in Fig. 4(b), making the number of components within subsystems as uniform as possible. In contrast to the previous subsystems in Fig. 4(a), all subsystems in Fig. 4(b) consist of the same number of components, so the learning rate of the agent is expected to be similar. The needs for this procedure are discussed through numerical

Table 2
Modularity and number of clusters according to edge elimination.

No. of eliminated edge	0	1	3	7	11	13	15	18
Modularity	0	0.443	0.565	0.685	**0.689**	0.673	0.647	0.608
No. of clusters	1	2	4	6	**8**	10	12	14

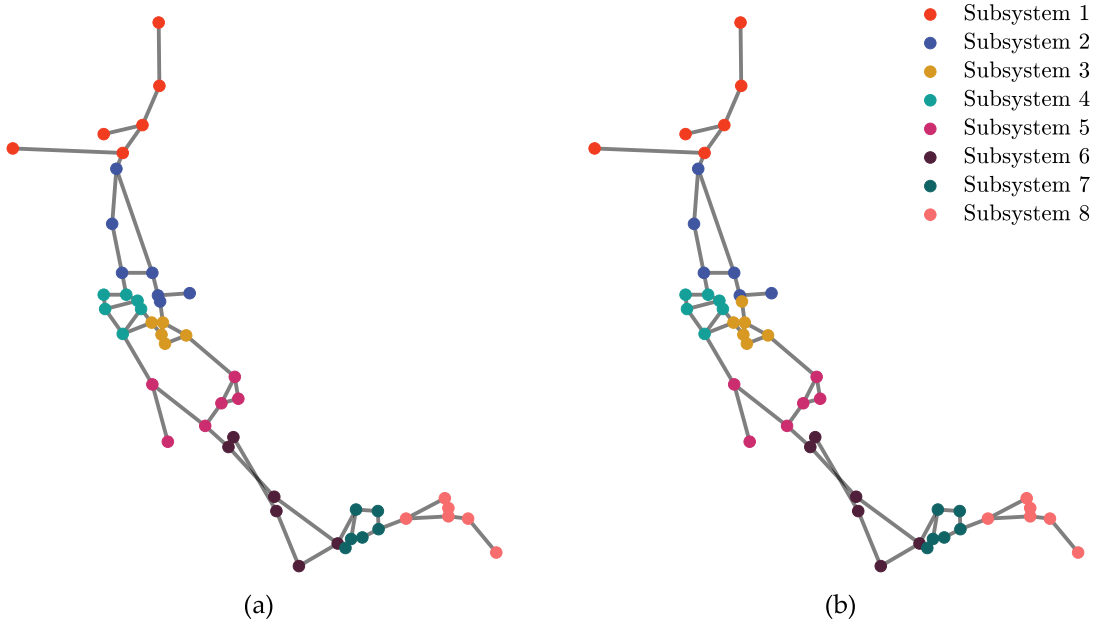


Fig. 4. (a) Subsystems detected by Girvan-Newman algorithm, and (b) subsystems with uniformly reallocated components.

Algorithm 1
Parallelized Multi-agent Deep Q-Network (PM-DQN).

Identify m subsystems through community detection and assign an agent to each subsystem
Initialize replay buffer D^j for $j \in [1,m]$
Initialize the online network Q_o^j with random parameters θ_j for $j \in [1,m]$
Initialize the target network Q_t^j with random parameters $\theta_j^- = \theta_j$ for $j \in [1,m]$
Initialize hyperparameters ω
for cycle $= 1$ **to** n_{cyc} **do**
  for processing unit $= 1$ **to** n_{pu} **do**
    for epoch $= 1$ **to** n_{epoch} **do**
      Initialize state $\boldsymbol{S}_t = [\boldsymbol{S}_t^j]_{j\in[1,n]}$ where $\boldsymbol{S}_t^j = [s_t^i]_{i\in cl_j}$
      for $t = 1$ **to** T_H **do**
        for $j = 1$ **to** m **do**
          Select action $a_t^j = \begin{cases} \text{a random action}, & \varepsilon \\ \text{argmin}_a Q_o^j(\boldsymbol{S}_t^j, a; \theta_j), & \text{otherwise} \end{cases}$
          Observe $\boldsymbol{S}_{t+1}^j$, $c_{comp,i}$ for $i \in sub_j$
        end
        Observe system damage cost c_{SD}
        for $j = 1$ **to** m **do**
          Calculate the factorized cost for subsystem j, c_{sub_j}
          Store experience $e_t^j = (\boldsymbol{S}_t^j, a_t^j, c_{sub_j}, \boldsymbol{S}_{t+1}^j)$ in D^j with TD-error δ_t^j
          Sample minibatch of experiences e_k^j with probability $|\delta_k^j|^\alpha / \sum_t |\delta_k^j|^\alpha$ from D^j
          Compute importance-sampling weight $w_k^j = (N \cdot P(e_k^j))^{-\beta}$
          Compute target value $y_k^j = c_{sub_j} + \gamma Q_t^j\left(\boldsymbol{S}_{k+1}^j, \underset{a\in A}{\text{argmin}} Q_o^j(\boldsymbol{S}_k^j, a; \theta_j); \theta_j^-\right)$
          Update $\delta_k^j \leftarrow y_k^j - Q_o^j(\boldsymbol{S}_k^j, a_k^j; \theta_j)$
          Perform gradient descent on $(\delta_k^j)^2$ w.r.t. θ_j
          Update $\theta_j^- \leftarrow \theta_j$ every C steps
        end
      end
    end
  end
  Synchronize the network parameters
  Tune hyperparameters $\omega \leftarrow \text{argmin}_\omega [Q^{sys}] + k\cdot\lambda$ for $k \in [-2, 2]$
end

examples in Section 4 by comparing the learning efficiency with or without the reallocation.

3.2. Step 2: factorization of centralized costs

After assigning one agent to each subsystem detected in Step 1, multiple agents make O&M decisions by comprehensively considering the state of components within each subsystem. To carry out the decision making independently but efficiently for all subsystems, the total cost c_{tot} should be factorized into individual subsystems based on effective multi-agent credit allocation. For example, the total cost c_{tot} in Eq. (1) is divided into two parts: decentralized costs (e.g., $c_{M,i}$ and $c_{S,i}$) and centralized costs (e.g., c_{SD}). Since decentralized costs obviously arise from each component, it makes sense to impose the sum of decentralized costs of the components in a subsystem to the corresponding agent. However, allocating the centralized cost to individual agents requires the introduction of VDN or predefined functions. To this end, PM-DQN finds the subsystems that are presumed to cause the loss of system, and selectively allocates centralized costs to those subsystems based on a predefined function. More specifically, by splitting the centralized cost and summing up the decentralized costs, the total decentralized cost c_{sub_j} for subsystem j, sub_j, is expressed as

$$c_{sub_j}(\boldsymbol{S}_t, a_t^j) = \sum_{i\in sub_j} \left[c_{M,i}(a_t^j) + c_{S,i}(s_t^i)\right] + \omega c_{f,j}(\boldsymbol{S}_t), \tag{14}$$

where a_t^j is redefined as the action chosen by agent j at time t; $c_{f,j}$ is the factorized cost transferred from the centralized cost c_{SD} to subsystem j; and ω is the hyperparameter that determines the weight of the factorized cost. The factorized cost $c_{f,j}$ is predefined as follows:

$$c_{f,j}(\boldsymbol{S}_t) = f_{SD}\cdot\min\left[L_{sys}(\boldsymbol{S}_t), L_j(\boldsymbol{S}_t^j)\right], \tag{15}$$

where $L_j(\cdot)$ is the QoS loss in the j^{th} subsystem; and $\boldsymbol{S}_t^j$ is a subset of $\boldsymbol{S}_t$, which is the state vector of components belonging to the j^{th} subsystem. While L_{sys} is calculated based on the maximum flow capacity between two predetermined terminals, L_j is defined as the difference between the total inflow and outflow of the j^{th} subsystem. The structure of Eq. (15) is similar to that of Eq. (2) before factorization, but individual agents infer the causality between QoS losses of system and their subsystems using the min operator.

Using Eqs. (6), (7), (14), and (15), one can calculate individual Q-values $Q^j(\boldsymbol{S}_t, a_t^j)$ for each agent j. In this paper, we utilize the DDQN combined with PER (introduced in Section 2.3) for more efficient and stable Q-learning. More specifically, the online network with parameters θ_j and the target network with parameters θ_j^- for $j \in [1,m]$ take the system state vector $\boldsymbol{S}_t$ and the current time step t as inputs, and respectively output the Q-values Q_o^j and Q_t^j according to each action in the form of vectors. Then, the optimal action based on the estimated Q_o^j is converted into an $|A^j|$-dimensional one-hot encoding vector, which is multiplied with the vector form of Q_t^j to update the online Q-value. Unlike these online parameters θ_j that are updated every step, the target network parameters θ_j^- are periodically updated to the online parameters θ_j every C steps. It should be noted that, unlike VDN, the expected total life-cycle cost of the system Q^{sys} is not equal to the sum of those caused by the factorized costs $c_{f,j}$ [37]. As all agents choose the actions that minimize their respective Q^j, the system-level action set $\boldsymbol{A}_t$ is redefined as

$$\boldsymbol{A}_t = \left[\underset{a}{\text{argmin}} Q^j(\boldsymbol{S}_t, a)\right]_{j\in[1,m]}. \tag{16}$$

In the learning process of DDQN, the performance of the PM-DQN depends significantly on the hyperparameterω in Eq. (14). Therefore, the hyperparameter value should be appropriately determined depending on the environment (e.g., system topologies, O&M costs, discount factor γ). In addition, there is no way to find the optimal value analytically, while heuristics require substantial computational costs for $j \in [1,m]$.

3.3. Step 3: parallel processing-based hyperparameter tuning

For effective hyperparameter tuning, we introduce parallel processing in the proposed algorithm. Processing units (e.g., CPUs or GPUs) are classified into five groups with different hyperparameter values. In each unit, agents explore the optimal decentralized policies using the ε-greedy algorithm simultaneously under each given hyperparameter ω, as illustrated in Fig. 2(a). After training agents for a sufficiently large number of epochs, called a cycle, one can judge the superiority of policies based on hyperparameter values through the expected total life-cycle costs Q^{sys}. Prior to the next cycle, the parameters of the online and target networks in all processing units, θ_j and θ_j^- for $j \in [1,m]$, are synchronized with those showing the best performance in the previous cycle, as shown in Fig. 2(b), thereby contributing to improving the master policy. Through the synchronization process, some rarely explored near-optimal policies can be propagated to other processing units, resulting in significant performance gains. Then, hyperparameter ω is tuned as follows:

$$\omega \leftarrow \text{argmin}_\omega [Q^{sys}] + k\lambda, \tag{17}$$

where λ is an exponentially decaying step size; and $k \in [-2, 2]$. In addition, when the expected life-cycle cost has sufficiently converged through comparison of results of the parallel processing, the algorithm is terminated early. Algorithm 1 provides a pseudo-code of the proposed

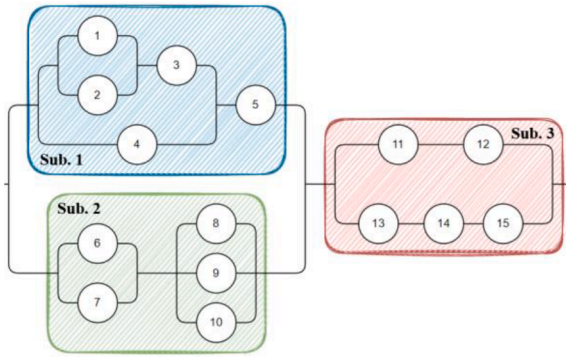


Fig. 5. A general system with 15 multi-state components, simplified by three subsystems.

Table 3
Expected life-cycle costs and 95% confidence intervals of realization of the trained policies with and without periodic synchronization in PM-DQN.

Number of Epochs	w/ Periodic Sync	w/o Periodic Sync
1,000	46.04±35.99	
2,000	33.65±3.44	
4,000	29.05±0.48	
6,000	28.71±0.53	
10,000	28.33±0.54	40.99±20.20

PM-DQN algorithm.

4. Numerical examples

To demonstrate the proposed PM-DQN, we consider two numerical examples inspired by Andriotis & Papakonstantinou [10] and Stern et al. [53]: (1) a multi-state lifeline network system with 15 components and (2) the simplified California gas distribution system abovementioned. The life-cycle span of each system T_{LS} is set to 50 steps (i.e., 50 years) with a discount factor of $\gamma = 0.95$. The coefficient f_{SD} in Eqs. (2) and (15) for the system damage cost c_{SD} is set to 5.0 in both examples.

To build and implement our own custom environments, we use OpenAI Gym [54], a toolkit for reinforcement learning research. All experiments are performed using the Keras deep learning Python library [55] with Tensorflow backend [56] on a server with 2 Intel(R) Xeon(R) CPU Gold 6240R CPUs at 2.40 GHz and 256GB RAM. The online and target networks consist of three fully connected hidden layers, each with SELU activation functions per example. For stochastic gradient descent on the network parameter space, we use the Nesterov-accelerated Adaptive Moment Estimation (Nadam) optimizer [57], combining the Adaptive Movement Estimation (Adam) with Nesterov momentum [58]. DDQNs explore and exploit complex environments using the ε-greedy algorithm mentioned in Section 2.3, where the exploration probability ε decreases from 0.5 to 0 for every cycle along with the cosine function [59]. For model training based on parallel processing on multiple processors, the multiprocessing library [60] is implemented. Appendix B provides more detailed information about hyperparameters for PM-DQN.

Since it is intractable to obtain the optimal policy for these examples due to the curse of dimensionality, two conventional O&M schemes and two MARL methods are also implemented as baseline policies as follows to confirm the superior performance of the PM-DQN:

- Condition-based maintenance (CBM): agents repair components that have deteriorated below optimized threshold states. Since the number of all combinable threshold cases is exponentially proportional to the number of components n, it is extremely time-consuming to evaluate the system life-time cost for all policies. In numerical examples, the optimal threshold state set is explored through iterations that sequentially update the optimal threshold state of each component that minimizes the system life-cycle cost. The number of threshold combinations per iteration scales linearly with the number of components.
- Time-based maintenance (TBM): agents periodically repair individual components, regardless of their state, at certain time intervals optimized for each component. To find the optimal repair intervals, we use iterations in the same form as CBM's policy exploration.
- Subsystem-level optimal maintenance (SOM): agents assigned to subsystems independently learn policies to minimize the cost of each subsystem in Eq. (14), where the factorized centralized cost $c_{f,j}(\boldsymbol{S}_t)$ depends only on the QoS loss in the j^{th} subsystem, $c_{f,j}^{SOM}(\boldsymbol{S}_t^j) = f_{SD} \cdot L_j(\boldsymbol{S}_t^j)$, instead of Eq. (15). Other than the factorized cost, the identified subsystems to which agents are assigned and the hyperparameters in Appendix B are shared with PM-DQN.
- Deep Centralized Multi-agent Actor Critic (DCMAC) [10]: agents learn policies based on two separate neural networks, called actor-critic methods, each approximating a centralized policy function and the expected total life-cycle costs. Actions on individual components are learned conditionally independent of each other. The details of structures and hyperparameters tuned for DCMAC in each numerical examples are given in Appendix B. In Example 2, to shorten the training time and streamline the process of efficiently exploring learning rates for actor and critic networks, parallelized DCMAC is introduced with periodic synchronization and compared to single-processing DCMAC.

4.1. Example 1: multi-state general system with 15 components

A 15-component system is represented as a *general* system, i.e., not series- or parallel-system, in Fig. 5. All components are assumed to be Type I. The system QoS is defined as the maximum flow capacity MF_{sys} between both left and right sides, which is 2.0 when all components operate in the AGAN state. As indicated by the hatched blocks in Fig. 5, three subsystems of five components are detected based on the Girvan-Newman algorithm. By grouping components, the size of the action space is shrunk from $2^{15} = 32,768$ to $3 \times 2^5 = 96$, since the three agents only share information about the system state $\boldsymbol{S}_t$ and select actions a_t^j for $j \in [1, 3]$ independently.

Before comparing the results of the proposed algorithm with those of the four baseline policies, we discuss the appropriateness of periodic synchronization in the PM-DQN. Table 3 shows the average life-cycle costs and 95% confidence intervals of the realization of the policies trained on parallel units with and without periodic synchronization in the PM-DQN. The mean converges rapidly to the optimal life-cycle cost when accompanied by periodic synchronization. The standard deviation also decreases significantly, and the coefficient of variation (c.o.v.) is less than 1% after the fourth cycle (i.e., 4,000 epochs), at which hyperparameter ω is tuned around the optimal value of 4.75. Since the step size $\lambda = 0.25$ is already small enough, the expected life-cycle cost of the policy explored at this time does not have a large difference from the optimal life-cycle cost after the 10th cycle. That is, when the step size and c.o.v. are sufficiently small (e.g., less than 0.5 and 1%, respectively), the algorithm can be terminated early without significant loss in terms of performance compared to when it progressed to the end. On the other hand, with single-cycle learning, although learning through the same number of epochs, the life-cycle cost estimates have a significant variance, and the mean cost is also higher than that of periodic synchronization. This inferior performance arises because there is no chance to tune the hyperparameter ω and explore good policies in many ways. Some outliers inevitably occur during parallel processing, but there are no other means for correcting them. Conversely, even if one or two learners find a rare near-optimal policy, these policies are not

Table 4
Life-cycle costs and computational time of PM-DQN and four baselines for Example 1.

Method	Total life-cycle cost	Time (*sec*)
CBM	28.42	9,323
TBM	56.71	99,174
SOM	35.41	2,523
DCMAC (single processing)	28.57	125,773
PM-DQN	**28.33**	**2,647**

propagated to most computational resources and are not further developed.

In this example, $5 \times 15 = 75$ and $50 \times 15 = 750$ combinations of threshold state and time interval sets are explored to optimize CBM and TBM, respectively, through five iterations. As a result, the optimal threshold state set for CBM and the repair interval set for TBM are determined, as shown in Appendix C. In contrast, since MARL methods including SOM, PM-DQN, and DCMAC take the state combination of the components in a subsystem or a system or centralized costs as input, it is difficult to specify the damage state at which agents repair individual components. The performances of baselines and the proposed policy are estimated through 1,000 demonstrations in terms of the total life-cycle costs c_{tot}, and the results are summarized in Table 4 with the time required for iterative operation or model training. The TBM policy was evaluated as the most ineffective one due to the stochastic environment. Although the SOM shares the subsystems and hyperparameters for model learning with PM-DQN, each agent learns the subsystem-level optimal policy, resulting in higher life-cycle costs than PM-DQN at the system level. On the other hand, the optimal policy learned by the PM-DQN shows the lowest estimated life-cycle cost, which is almost identical to those of the optimal policies proposed by CBM and DCMAC. Comparing the computational time among them, the advantage of PM-DQN in computational efficiency becomes clear. CBM takes 3.5 times longer than the computational time of PM-DQN, even though search of all combinations is replaced with iterations. Although DCMAC requires about 47.5x more learning time than PM-DQN due to the difference in the number of processors, it can be shortened through parallelization

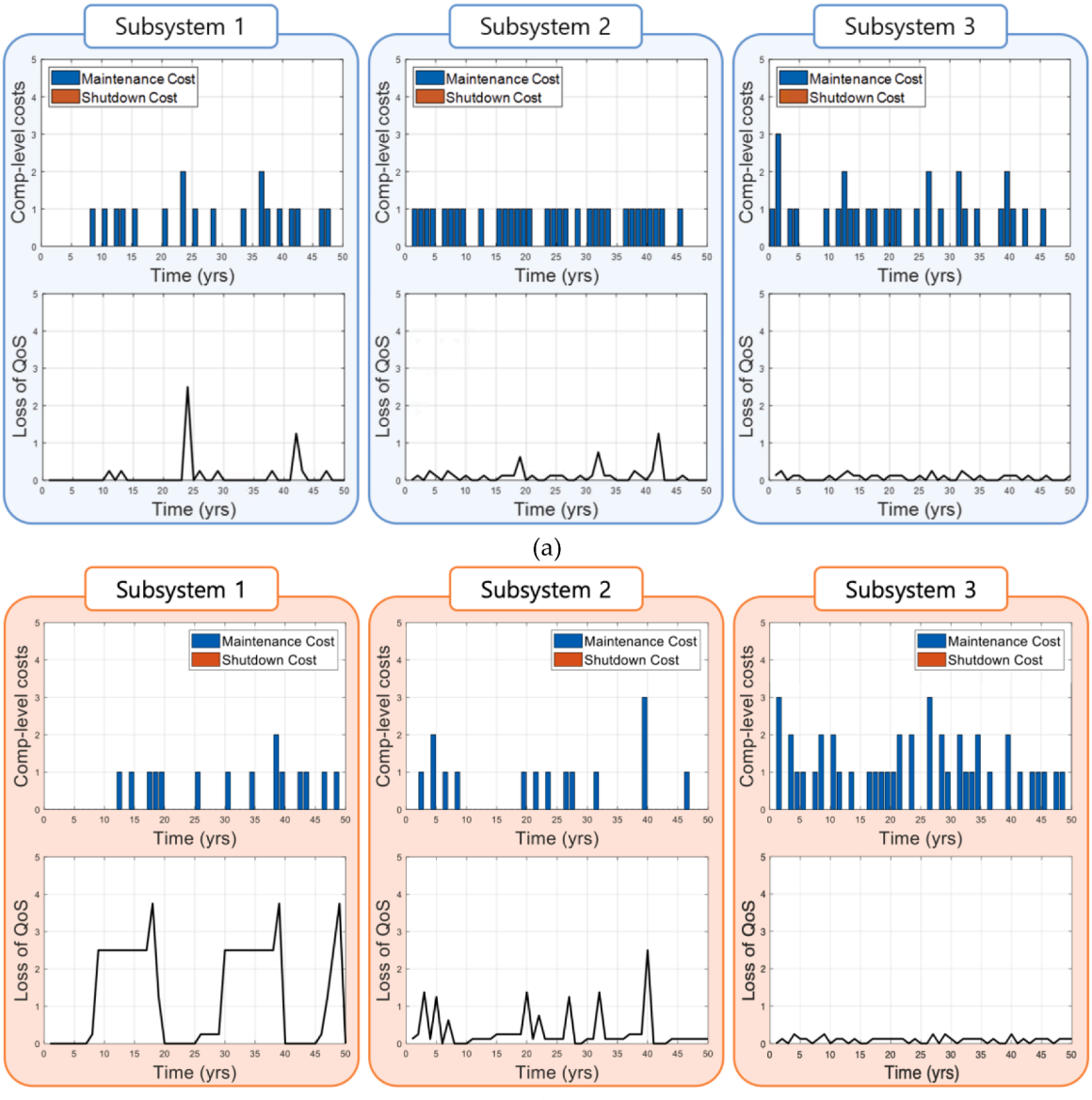


Fig. 6. Life-cycle O&M results during an epoch in Example 1: component-level costs (i.e., $c_{M,i}$ and $c_{S,i}$) and loss of QoS in each subsystem (i.e., $c_{f,j}^{SOM} = f_{SD} \cdot L_j$) under (a) the SOM policy, and (b) the PM-DQN policy.

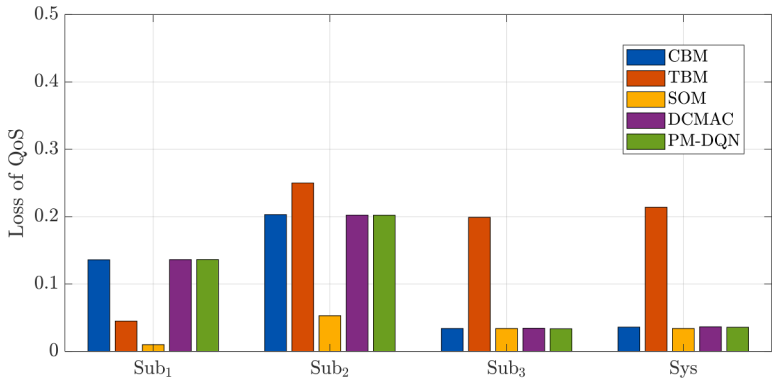


Fig. 7. Annual average loss of QoS in individual subsystems or system in Example 1 under each policy.

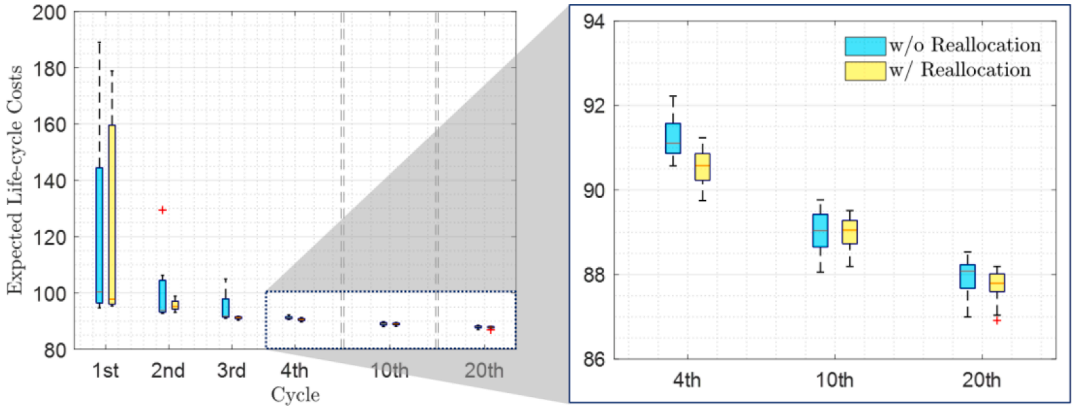


Fig. 8. Expected life-cycle costs for learning cycles with and without reallocation in PM-DQN.

Table 5
Life-cycle costs and computational time of PM-DQN and four baselines for Example 2.

Method	Total life-cycle cost	Time (*sec*)
CBM	89.95	20,126
TBM	161.61	214,140
SOM	88.82	103,456
DCMAC (parallel processing)	87.13	137,211
PM-DQN (w/o Reallocation)	86.77	97,437
PM-DQN (w/ Reallocation)	**86.60**	**101,934**

and the improved convergence speed is compared on the complex system in Example 2.

Fig. 6(a) and (b) show the realization of the O&M process for the individual subsystems according to the optimal policies based on SOM and PM-DQN, respectively. In these figures showing time histories of component-level costs in each subsystem, blue bars indicate maintenance costs $c_{M,i}$ spent on repairs, while red bars, representing shutdown costs $c_{S,i}$, are not marked on the chart because all components are repaired before shutdown. Under the SOM policy, each agent immediately takes the O&M policy to minimize the loss of QoS in individual subsystems, thereby repairing subsystems immediately even minor damage at the subsystem level as if they were part of a series system as shown in Fig. 6(a). In the PM-DQN policy, the agent managing subsystem 3 operates in the same way as in the SOM policy, because subsystem 3 significantly influences the system QoS. In contrast, the other two subsystems connected in parallel are treated even more tolerantly than the SOM, since the two act as a detour to each other and the failure of either subsystem does not directly lead to the loss of system QoS. This difference between these two algorithms is clearly shown in Fig. 7, which illustrates the annual average flow capacity loss in individual subsystems and the system under all policies. As can be seen from the results of all policies, the QoS loss in subsystem 3 plays a dominant role in the QoS loss of the system due to its series-connected topology in the system.

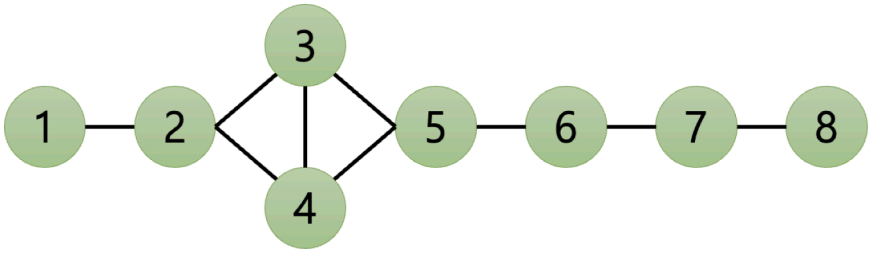


Fig. 9. Simplified California gas distribution system consisting of subsystems.

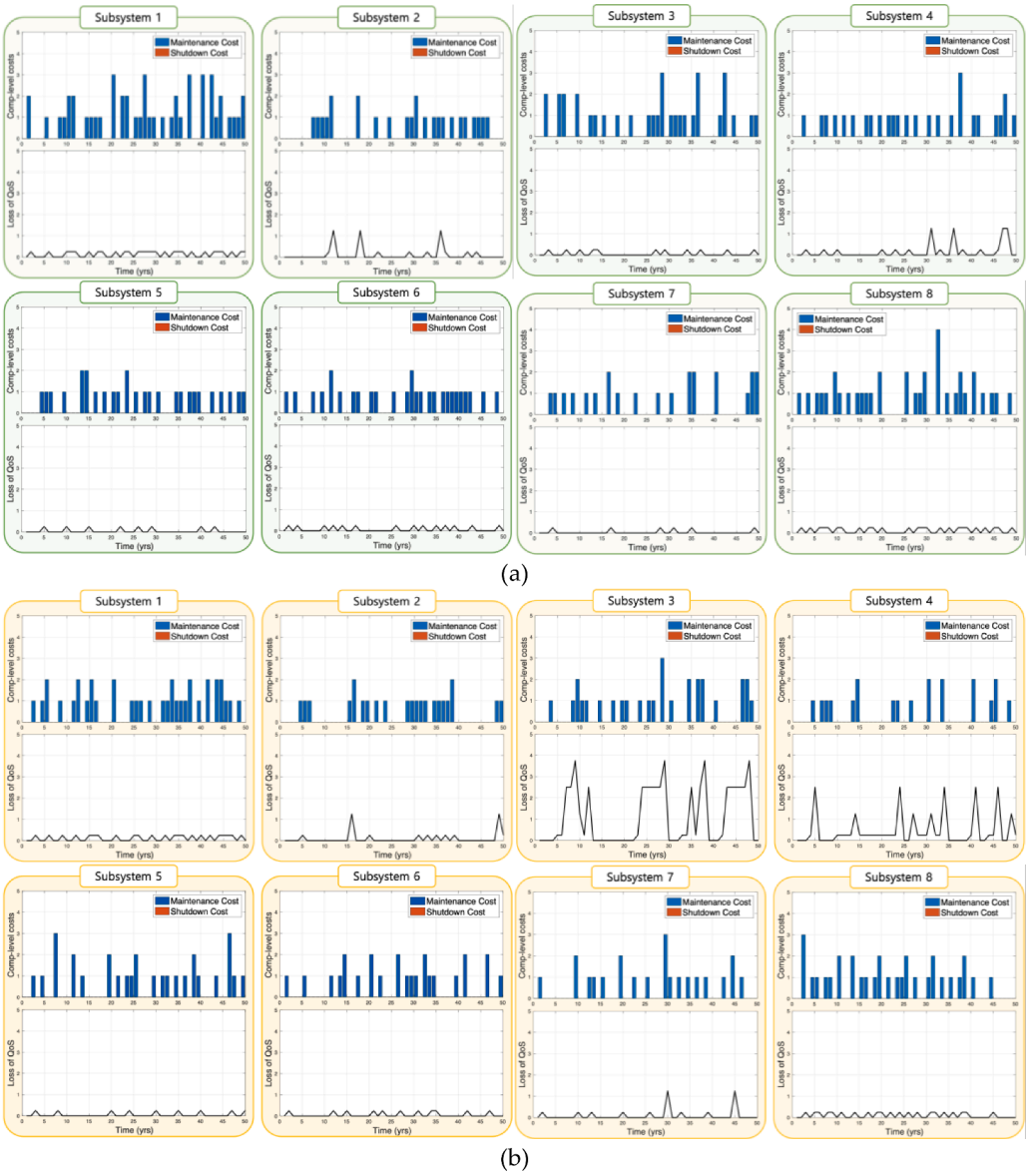


Fig. 10. Life-cycle O&M results during an epoch in Example 2: component-level costs (i.e., $c_{M,i}$ and $c_{S,i}$) and loss of QoS in each subsystem (i.e., $c_{f,j}^{SOM} = f_{SD} \cdot L_j$) under (a) the SOM policy, and (b) the PM-DQN policy.

4.2. Example 2: the simplified California gas distribution system with 48 gas substations

Example 2 deals with a more realistic lifeline system, the California gas distribution system in Fig. 3. Unlike Example 1, there are two types of components in the system; some components located in the west, close to the Pacific Ocean, are modeled as type II components with a high deterioration rate, while the rest are modeled as Type I. The maximum flow capacity MF_{sys} between two terminal nodes is given as 1.0 when all substations operate in the AGAN state. Through community detection shown in Fig. 4(a), the size of the action space is shrunk from $2^{48} \cong 2.81 \times 10^{14}$ to $2^5 + 6 \times 2^6 + 2^7 = 544$. After resizing the subsystems as shown in Fig. 4(b), the largest action space is reduced from $2^7 = 128$ to $2^6 = 64$.

Before comparing the performance of the proposed algorithm with the baselines, we examine the needs for component reallocation discussed in Section 3.1. Boxplots in Fig. 8 show the expected life-cycle costs before and after reallocating the subsystem sizes evenly for learning cycles of PM-DQN. Outliers under the worst performing hyperparameters are excluded for visualization purposes. Comparing the training results up to the third cycle, in which sufficient learning has not been achieved yet, the variance of the learning rate of PM-DQN without reallocation is significantly larger than that with reallocation. This is because the policy search for a relatively large number of state and action spaces has not been conducted effectively and sufficiently in the largest subsystem 2. While their variances become similar after the 4th cycle, the former's estimated life-cycle cost becomes equal to that of the latter only after the 10th cycle is completed. As learning continues, both expected costs keep decreasing, achieving almost the same life-cycle cost. This shows that PM-DQN works well for large systems with components that have different degradation processes, although the convergence rate becomes lower than before.

The performances of baselines and MARL-based policies including the PM-DQN without reallocation estimated through 1,000 demonstrations are summarized in Table 5. The optimal CBM and TBM policies are explored through five iterations as in Example 1. As a result, the threshold state set and repair time interval set are respectively determined, as shown Appendix C. As in Example 1, the TBM policy shows

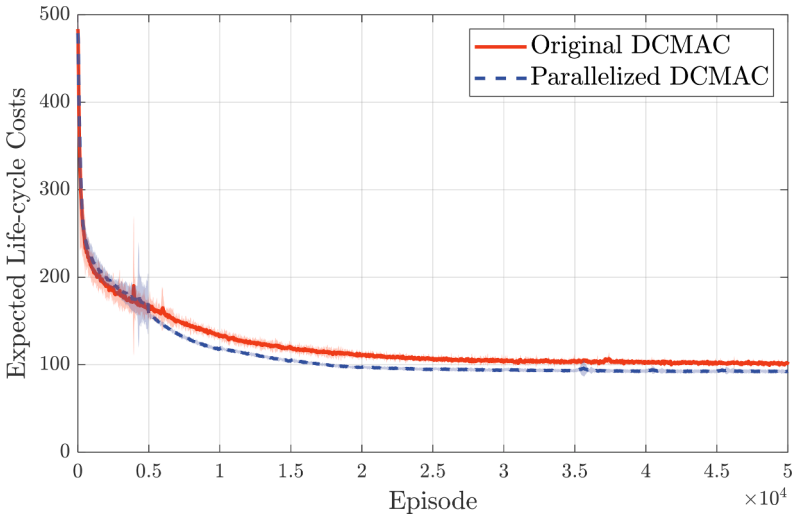


Fig. 11. Training history of original and parallelized DCMAC with 95% confidence intervals in Example 2.

inferior results due to the stochastic environment, and the life-cycle cost of the CBM policy, showing excellent performance in Example 1, is higher than those of other MARL-based algorithms. While CBM evaluates the effect of individual components on system QoS at the component level and determines the state thresholds for repair, the system QoS fundamentally depends on the combination of states of various components. This characteristic becomes more prominent as systems get more complex, which is confirmed through the performance change of CBM compared to MARL-based approaches in Example 1 and Example 2. Although the CBM policy ranks with the DCMAC and PM-DQN policies in a simple system in Example 1, it faces the limitations in Example 2.

This also applies to SOM. Although the SOM policy may be an optimal decision at each subsystem level, the result of the optimal PM-DQN policy in Table 5 shows that the subsystem-level optimal policy does not match that at the system level, where the hyperparameter ω is eventually tuned to 23.90. Fig. 9 shows a simplified California gas system consisting of 9 subsystems, and the realizations of the system O&M by SOM and PM-DQN policies are shown in Fig. 10(a) and (b), respectively. To minimize the QoS loss at the subsystem level, SOM manages subsystems 3 and 4 conservatively compared to PM-DQN even though they are connected in parallel. Accordingly, the loss of QoS in individual subsystems is well maintained at a low level in the SOM policy as shown in Fig. 10(a). However, because large maintenance cost is required for the SOM policy, the life-cycle cost under the policy shows inferior results to the PM-DQN and DCMAC policies.

In contrast, the PM-DQN policy considers the QoS loss of subsystems and systems simultaneously. Subsystems except for subsystems 3 and 4 have no separate bypass. This means that, even if agents maintain subsystems 3 and 4 loosely, other subsystems should be managed conservatively as shown in Fig. 10(b). In this way, system simplification based on community detection identifies the system topology and provides the basis for convergence of each agent's local optimal policy to the global optimal policy. The PM-DQN policy takes advantage of this to manage the system, thereby showing the best expected life-cycle costs at the system level.

Fig. 11 shows the training history of DCMAC with and without parallel processing. In parallel processing, the optimal learning rates for actor and critical networks are explored in the range of $[10^{-7}, 10^{-3}]$ and $[10^{-5}, 10^{-3}]$ respectively, gradually converging to the combination of learning rates with the lowest life-cycle costs. The best learning rates among the explored ones are utilized for single-processing DCMAC as well as the parallelized DCMAC. For the first cycle (i.e., the first 5,000 episodes), there is not much difference between the two results. However, the gap widens considerably right after the synchronization of parallel-trained policies. The difference never diminishes until the end of training, and the superior performance resulting from parallelization is comparable to PM-DQN; comparing them in terms of performance and computation time, PM-DQN is slightly better, but the difference is not significant considering the structural difference between the two models. This implies that the proposed parallel processing framework with periodic synchronization has a dominant effect on performance improvement and it can be transferred to other DRL methods.

5. Summary and conclusions

This paper proposed an optimal decision-making framework based on deep reinforcement learning (DRL), termed parallelized multi-agent deep Q-network (PM-DQN), for efficient risk-informed operation and management (O&M) scheduling of large civil lifeline systems. Existing methods dealt with these system-level O&M scheduling problems by limiting the size of the systems or approximating them as component-level subproblems due to the exponentially growing state and action spaces. However, these types of computational complexity reduction may incur a significant loss of accuracy. Moreover, even if the action space is relaxed, it is still necessary to find the best policy by exploring all policy combinations. To find the optimal policy, the multi-agent credit allocation problem should be solved by inferring the contributions to the centralized cost function. Unstable feedback depending on the policy selection of other agents hinders decentralized policy learning. In contrast, the proposed algorithm overcame this challenge by introducing a divide-and-conquer strategy with community detection and parallel processing. Since network clustering is based on the system topology, the subsystems identified by the Girvan-Newman clustering algorithm can achieve an appropriate balance between accuracy loss and computational complexity reduction. The strength of the proposed algorithm is further enhanced by reducing the policy exploration time and tuning the optimal hyperparameters in combination with parallel operation. Multiple processing units derive an optimal policy set under hyperparameter tuning and periodic synchronization with the best one. The accuracy and efficiency of PM-DQN were demonstrated on a multi-state general system with 15 components and the California gas distribution system with 48 components. In each numerical example, the optimal PM-DQN policies outperformed baseline alternatives including conventional O&M policies and other MARL-based policies in terms of computational time as well as the expected life-cycle costs. In particular, the California gas distribution system represented as a general system of subsystems through community detection shows the system's topological characteristics prominently, indicating the reason for the good

performance of PM-DQN in subsystem level decision-making.

The proposed PM-DQN trains multiple agents simultaneously by defining factorization cost functions based on the causal relationship between the flow capacity losses in the whole system and subsystems. Because the hyperparameter tuning and the entire process of policy exploration are performed independently across multiple processing units, the proposed algorithm has high scalability for the ever-evolving scale of processing units. In addition, the algorithm features flexible handling of the problem, such as adjusting the number of processing units according to the complexity of problems and the computing power. The parallelization remains scalable and flexible even when other MARL algorithms are used. However, there are some other obstacles arising from the limitations of MDPs that hinder the modeling and application of real lifeline network system O&M; it is difficult to consider the time required for actions in the discrete-time domain, and there is a time gap between state change and maintenance actions. In addition, if more diverse action options, such as minor repairs or reinforcements, are given, policy learning at the subsystem level may suffer from the curse of dimensionality. Nevertheless, for efficient MARL and parallel processing, the proposed framework of deploying agents and periodically synchronizing multiple processors can be applied to various other DRL methods. As shown in Example 2, vanilla DRLs specialized for parallel processing are expected to improve exploration and training stability, thereby enabling simple but powerful end-to-end learning without additional clustering or cost splitting.

Furthermore, we can extend the proposed method to a partially observable MDP (POMDP) environment by considering monitoring errors in grasping system states. The computational cost is higher than that in MDP environment because belief function is continuously updated, and monitoring action is additionally considered. However, we can broaden the scalability of the existing algorithms by taking advantage of parallel processing and system simplification based on network clustering. Further research is underway to model state changes in the continuous-time domain and actions that take different times using Semi-Markov decision processes [23]. In addition, the development of such a framework is expected to achieve the scalability to cope with unexpected events (e.g., earthquakes, typhoons).

CRediT authorship contribution statement

Dongkyu Lee: Visualization, Validation, Software, Methodology, Conceptualization, Investigation, Writing – original draft. **Junho Song:** Resources, Project administration, Funding acquisition, Conceptualization, Supervision, Writing – review & editing.

Declaration of Competing 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.

Data availability

Data will be made available on request.

Acknowledgement

This first author is supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2022–00144434). The corresponding author is supported by the Institute of Construction and Environmental Engineering at Seoul National University. These supports are gratefully acknowledged.

Appendix A. State-transition matrix

A1. State-transition matrix of component type I for '*DN*'

$$T_{DN}^{I} = \begin{bmatrix} 0.8 & 0.2 & 0 & 0 & 0 \\ 0 & 0.8 & 0.2 & 0 & 0 \\ 0 & 0 & 0.8 & 0.2 & 0 \\ 0 & 0 & 0 & 0.8 & 0.2 \\ 0 & 0 & 0 & 0 & 1.0 \end{bmatrix}$$

A2. State-transition matrix of component type II for '*DN*'

$$T_{DN}^{II} = \begin{bmatrix} 0.7 & 0.3 & 0 & 0 & 0 \\ 0 & 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0.7 & 0.3 & 0 \\ 0 & 0 & 0 & 0.7 & 0.3 \\ 0 & 0 & 0 & 0 & 1.0 \end{bmatrix}$$

A3. State-transition matrix for '*R*'

$$T_{R}^{I} = T_{R}^{II} = \begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Appendix B. Details of PM-DQN and DCMAC for Examples 1 & 2

Table B1–B2

Table B1
PM-DQN hyperparameters.

Hyperparameter	Example 1	Example 2
Experience replay size	50,000	50,000
Exploration probability ε	$0.5 \rightarrow 0$	$0.5 \rightarrow 0$
Frequency of updating the target network $\boldsymbol{C}$	50	50
Initial range of ω	[0,8]	[0,32]
Nadam learning rate	0.001	0.001
Number of cycles n_{cyc}	10	20
Number of epochs per cycle n_{epoch}	1,000	3,000
Number of subsystems (or agents)	3	8
Number of input layer nodes (per agent)	5	6
Number of output layer nodes (per agent)	32	64
Number of hidden layer nodes (per agent)	(32, 32, 32)	(64, 64, 64)
Number of used process units n_{pu}	20	35
PER importance sampling α	0.6	0.6
PER bias correction β	0.4	0.4
Size of minibatch	64	128

Table B2
Details of DCMAC.

Hyperparameter/Architecture	Example 1	Example 2
Number of used process units n_{pu}	1	36
Number of epochs per cycle n_{epoch}	–	5,000
Number of cycles n_{cyc}	–	10
Batch size	64	128
Buffer memory size	300,000	300,000
Clipping factor c for IS weights	2.0	2.0
Exploration probability ε	$0.5 \rightarrow 0.001$	$0.5 \rightarrow 0.0001$
Number of hidden layer nodes in actor networks	(32, 32, 32)	(64, 64, 64)
Optimizer for actor & critic networks	Nadam	Nadam
Nadam learning rate for actor networks	10^{-4}	$[10^{-7}, 10^{-3}]$
Nadam learning rate for critic network	10^{-3}	$[10^{-5}, 10^{-3}]$
Activation function in actor hidden layers	SELU	SELU
Activation function in critic hidden layers	SELU	SELU
Activation function in actor output layer	Softmax	Softmax
Activation function in critic output layer	Linear	Linear

Appendix C. Optimal threshold states for condition-based maintenance and time intervals for time-based maintenance

Table C1–C2

Table C1
Optimal threshold states for CBM in Examples 1 & 2.

	Threshold state of each component
Example 1	[3,3,3,3,3,2,2,3,3,3,1,1,1,1,1]
Example 2	[1,2,1,1,3,3,1,2,4,2,2,2,2,3,3,4,3,3,2,3,2,2,3,4, 3,1,2,3,2,3,2,3,4,2,2,2,3,4,4,1,2,2,1,2,3,1,1,1]

Table C2
Optimal time intervals for TBM in Examples 1 & 2.

	Repair interval of each component
Example 1	[15,15,15,10,10,10,15,15,15,15,5,5,5,5,5]
Example 2	[7,7,7,8,11,11,7,10,7,13,7,9,9,9,12,10,9,8,7,11,17,7,13,12, 7,7,9,7,17,9,11,13,9,10,11,7,15,13,11,7,10,17,7,13,11,8,7,8]

References

[1] Niu YF. Performance measure of a multi-state flow network under reliability and maintenance cost considerations. Reliab Eng Syst Saf 2021;215:107822.
[2] Dong Y, Frangopol DM. Risk-informed life-cycle optimum inspection and maintenance of ship structures considering corrosion and fatigue. Ocean Eng 2015; 101:161–71.
[3] Aryai V, Baji H, Mahmoodian M, Li CQ. Time-dependent finite element reliability assessment of cast-iron water pipes subjected to spatio-temporal correlated corrosion process. Reliab Eng Syst Saf 2020;197:106802.
[4] Morales-Torres A, Escuder-Bueno I, Serrano-Lombillo A, Rodríguez JTC. Dealing with epistemic uncertainty in risk-informed decision making for dam safety management. Reliab Eng Syst Saf 2019;191:106562.
[5] Ahmad R, Shahrul K. An overview of time-based and condition-based maintenance in industrial application. Comput Ind Eng 2012;63(1):135–49.
[6] de Jonge B, Teunter R, Tinga T. The influence of practical factors on the benefits of condition-based maintenance over time-based maintenance. Reliab Eng Syst Saf 2017;158:21–30.
[7] Prajapati A, Bechtel J, Ganesan S. Condition based maintenance: a survey. J Qual Maintenance Eng 2012.
[8] Omshi EM, Grall A. Replacement and imperfect repair of deteriorating system: study of a CBM policy and impact of repair efficiency. Reliab Eng Syst Saf 2021; 215:107905.
[9] Martínez-Galán Fernández P, Guillén López AJ, Márquez AC, Gomez Fernández JF, Marcos JA. Dynamic Risk Assessment for CBM-based adaptation of maintenance planning. Reliab Eng Syst Saf 2022;223:108359.

[10] Andriotis CP, Papakonstantinou KG. Managing engineering systems with large state and action spaces through deep reinforcement learning. Reliab Eng Syst Saf 2019;191:106483.
[11] Pages A, Gondran M, Griffin E. System reliability: evaluation & prediction in engineering. New York, NY: Springer; 1986.
[12] Lewis EE. Introduction to reliability engineering. 2nd ed. New York, NY: John Wiley & Sons; 1995.
[13] Der Kiureghian A, Ditlevsen OD, Song J. Availability, reliability and downtime of systems with repairable components. Reliab Eng Syst Saf 2007;92(2):231–42.
[14] Ouyang Y, Madanat S. An analytical solution for the finite-horizon pavement resurfacing planning problem. Transp Res Part B 2006;40(9):767–78.
[15] Compare M, Marelli P, Baraldi P, Zio E. A Markov decision process framework for optimal operation of monitored multi-state systems. In: Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability. 232; 2018. p. 677–89.
[16] Nielsen JS, Sørensen JD. Methods for risk-based planning of O&M of wind turbines. Energies 2014;7(10):6645–64.
[17] Ohlmann JW, Bean JC. Resource-constrained management of heterogeneous assets with stochastic deterioration. Eur J Oper Res 2009;199(1):198–208.
[18] Medury A, Madanat S. Incorporating network considerations into pavement management systems: a case for approximate dynamic programming. Transp Res Part C 2013;33:134–50.
[19] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing atari with deep reinforcement learning. arXiv preprint 2013. *arXiv:1312.5602*.
[20] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Hassabis D. Human-level control through deep reinforcement learning. Nature 2015;518 (7540):529–33.
[21] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Hassabis D. Mastering the game of Go with deep neural networks and tree search. Nature 2016; 529(7587):484–9.
[22] Watkins CJ, Q-learning DP. Mach Learn 1992;8(3):279–92.
[23] Papakonstantinou KG, Shinozuka M. Planning structural inspection and maintenance policies via dynamic programming and Markov processes. Part I: theory. Reliab Eng Syst Saf 2014;130:202–13.
[24] Papakonstantinou KG, Shinozuka M. Planning structural inspection and maintenance policies via dynamic programming and Markov processes. Part II: POMDP implementation. Reliab Eng Syst Saf 2014;130:214–24.
[25] Memarzadeh M, Pozzi M, Zico Kolter J. Optimal planning and learning in uncertain environments for the management of wind farms. J Comput Civ Eng 2015;29(5): 04014076.
[26] Yang A, Qiu Q, Zhu M, Cui L, Chen W, Chen J. Condition based maintenance strategy for redundant systems with arbitrary structures using improved reinforcement learning. Reliab Eng Syst Saf 2022:108643.
[27] Hasselt H. Double Q-learning. Adv Neural Inf Process Syst 2010:23.
[28] Zhang N, Si W. Deep reinforcement learning for condition-based maintenance planning of multi-component systems under dependent competing risks. Reliab Eng Syst Saf 2020;203:107094.
[29] Mohammadi R, He Q. A deep reinforcement learning approach for rail renewal and maintenance planning. Reliab Eng Syst Saf 2022:108615.
[30] Tan M. Multi-agent reinforcement learning: independent vs. cooperative agents. In: Proceedings of the tenth international conference on machine learning; 1993.
[31] Foerster J, Nardelli N, Farquhar G, Afouras T, Torr PH, Kohli P, Whiteson S. Stabilising experience replay for deep multi-agent reinforcement learning. In: International conference on machine learning; 2017.
[32] Sunehag P, Lever G, Gruslys A, Czarnecki WM, Zambaldi V, Jaderberg M, Graepel T. Value-decomposition networks for cooperative multi-agent learning. arXiv preprint 2017. *arXiv:1706.05296*.
[33] Gronauer S, Diepold K. Multi-agent deep reinforcement learning: a survey. Artif Intell Rev 2022;55:895–943.
[34] Andriotis CP, Papakonstantinou KG. Deep reinforcement learning driven inspection and maintenance planning under incomplete information and constraints. Reliab Eng Syst Saf 2021;212:107551.
[35] Zhou Y, Li B, Lin TR. Maintenance optimisation of multicomponent systems using hierarchical coordinated reinforcement learning. Reliab Eng Syst Saf 2022;217: 108078.
[36] Nguyen VT, Do P, Vosin A, Iung B. Artificial-intelligence-based maintenance decision-making and optimization for multi-state component systems. Reliab Eng Syst Saf 2022;228:108757.
[37] Rashid T, Samvelyan M, Schroeder C, Farquhar G, Foerster J, Qmix WS. Monotonic value function factorisation for deep multi-agent reinforcement learning. In: International Conference on Machine Learning; 2018.
[38] Lim HW, Song J, Kurtz N. Seismic reliability assessment of lifeline networks using clustering-based multi-scale approach. Earthquake Eng Struct Dyn 2015;44(3): 355–69.
[39] Lee D, Song J. Multi-scale seismic reliability assessment of networks by centrality-based selective recursive decomposition algorithm. Earthquake Eng Struct Dyn 2021;50(8):2174–94.
[40] Ahuja RK, Kodialam M, Mishra AK, Orlin JB. Computational investigations of maximum flow algorithms. Eur J Oper Res 1997;97(3):509–42.
[41] Lee YJ, Song J, Gardoni P, Lim HW. Post-hazard flow capacity of bridge transportation network considering structural deterioration of bridges. Struct Infrastruct Eng 2011;7(7–8):509–21.
[42] Choi E, Song J. Cost-effective retrofits of power grids based on critical cascading failure scenarios identified by multi-group non-dominated sorting genetic algorithm. Int J Disaster Risk Reduct 2020;49:101640.
[43] Sutton RS, Barto AG. Reinforcement learning: an introduction. Cambridge, MA: MIT press; 2018.
[44] Bellman R. Dynamic programming. Science 1996;153(3731):34–7.
[45] Lin LJ. Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach Learn 1992;8(3–4):293–321.
[46] Schaul T, Quan J, Antonoglou I, Silver D. Prioritized experience replay. arXiv preprint 2015. *arXiv:1511.05952*.
[47] Kok JR, Vlassis N. Collaborative multiagent reinforcement learning by payoff propagation. J *Mach Learn Res* 2006;7:1789–828.
[48] Von Luxburg U. A tutorial on spectral clustering. Stat Comput 2007;17(4): 395–416.
[49] van Dongen S. Ph.D. Thesis. University of Utrecht; 2000.
[50] Gomez C, Sanchez-Silva M, Dueñas-Osorio L, Rosowsky D. Hierarchical infrastructure network representation methods for risk-based decision-making. Struct Infrastruct Eng 2013;9(3):260–74.
[51] Newman ME, Girvan M. Finding and evaluating community structure in networks. Phys Rev E 2004;69(2):026113.
[52] Girvan M, Newman ME. Community structure in social and biological networks. Proc Natl Acad Sci 2002;99(12):7821–6.
[53] Stern RE, Song J, Work DB. Accelerated Monte Carlo system reliability analysis through machine-learning-based surrogate models of network connectivity. Reliab Eng Syst Saf 2017;164:1–9.
[54] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W. Openai gym. arXiv preprint 2016. *arXiv:1606.01540*.
[55] Chollet F. Keras. 2015. Available at: https://keras.io.
[56] Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Zheng X. TensorFlow: a system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation; 2016. p. 265–83.
[57] Dozat T. Incorporating nesterov momentum into adam. 2016.
[58] Nesterov Y. Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM J Optim 2012;22(2):341–62.
[59] Loshchilov I, Sgdr HF. Stochastic gradient descent with warm restarts. arXiv preprint 2016. *arXiv:1608.03983*.
[60] Python Software Foundation. Multiprocessing - Process-based parallelism. 2023. Available at: https://docs.python.org/3/library/multiprocessing.html.