Weighted-QMIX-based optimization for maintenance decision-making of multi-component systems

Van-Thai Nguyen, Phuc Do, Alexandre Voisin, and Benoit Iung

Université de Lorraine, CNRS, CRAN, F-54000 Nancy, France
van-thai.nguyen@univ-lorraine.fr
phuc.do@univ-lorraine.fr
alexandre.voisin@univ-lorraine.fr
benoit.iung@univ-lorraine.fr

ABSTRACT

It is well-known that maintenance decision optimization for multi-component systems faces the curse of dimensionality. Specifically, the number of decision variables needed to be optimized grows exponentially in the number of components causing computational expensive for optimization algorithms. To address this issue, we customize a multi-agent deep reinforcement learning algorithm, namely Weighted QMIX, in the case where system states can be fully observed to obtain cost-effective policies. A case study is conducted on a 13-component system to examine the effectiveness of the customized algorithm. The obtained results confirmed its performance.

1. INTRODUCTION

Maintenance policy can be classified into two main categories, namely, corrective and preventive maintenance (CM and PM) (H. Wang, 2002). CM carries out maintenance actions on failed machines, which is often associated with high related costs due to unexpected production losses as well as unscheduled maintenance costs (Ahmad & Kamaruddin, 2012). On the contrary, PM implements maintenance on functioning machines to prevent their sudden failures in order to reduce downtime costs (Huang, Chang, & Arinez, 2020).

PM interventions can be planned in either time-oriented or condition-based manner (CBM), however, the later appears to be more advantageous. Particularly, it allows flexibly selecting maintenance decisions based on current states of maintained machines instead of on a fixed scheduled calendar. Moreover, recent advances in sensing and information technology allows rich degradation data to be collected enabling CBM to become a popular approach for maintenance decision-making and optimization.

Van-Thai Nguyen et al. This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 United States License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

CBM policies can be divided into two main groups: direct mapping and threshold-based policy. While the former maps directly from component degradation measurements to maintenance actions, the later first compares component states to predefined thresholds, and then choose maintenance actions accordingly. Whereas CBM optimization processes used for single-unit systems can be effectively achieved due to the small number of decision variables needed to be optimized (Quatrini, Costantino, Di Gravio, & Patriarca, 2020), the ones of multi-component systems suffer from the curse of dimensionality. Particularly, the number of decision variables grows rapidly as the number of components increases, causing computational expensive for optimization algorithms (Zhang & Si, 2020).

Recent advancements in the field of reinforcement learning (RL) give rise to direct mapping approaches by providing new tools for single-agent deep RL algorithms (DRL) to deal with maintenance decision optimization of systems with large state spaces. Specifically, (Zhang & Si, 2020) used double deep Q-network (DDQN) algorithm to minimize cost for a 12-component system which suffers from stochastic, economic dependence and competing failure risks. DDQN is also employed to optimize maintenance cost for systems with extremely large state spaces showing better performance in comparison to threshold-based policies (Huang et al., 2020). Despite the success of single-agent DRL algorithms for maintenance applications with state space complexities, they are shown in the literature to suffer from the problem of large action spaces (Andriotis & Papakonstantinou, 2019). In particular, the output layer of a conventional deep Q-network is composed of Q-value for each available action, and is then equal to the action space's size. Similarly, actor networks of policy-gradient-based DRL algorithms output a probability distribution over all possible actions. As a result, these network structures of single-agent DRL algorithms are not suitable for applications with sizable action spaces.

Fortunately, the framework of multi-agent DRL (MADRL) appears as a promising solution to this challenge. Particularly, in a multi-agent maintenance planning task, each agent observes the state of a subsystem or the system and determines maintenance actions of one or more components. As a result, the action space of an agent is much smaller than the one of the entire system, which helps alleviate the issue of large action spaces at system level (Zhou, Li, & Lin, 2021). MADRL has been received recently increasing attention from maintenance researchers and several related articles are reviewed in the following. Wolpertinger deep deterministic policy gradient algorithm is employed in (Liu, Chen, & Jiang, 2019) to optimize selective maintenance policies for a coal transport system consisting of 14 components. However, this algorithm requires a nearest neighbor layer used for action reduction that interrupts the differentiability of the network, potentially leading to training instabilities due to improper backpropagation of gradients (Andriotis & Papakonstantinou, 2019). Deep centralized multi-agent actor-critic (DCMAC) algorithm is developed by (Andriotis & Papakonstantinou, 2019) to optimize maintenance actions for large structures. The truncated importance sampling mechanism is employed in DCMAC to cope with high variance in gradient estimators of learning policies, however, bias still exists which may cause unstable training (Z. Wang et al., 2016). More recently, hierarchical coordinated DRL algorithm is proposed in (Zhou et al., 2021) to optimize maintenance decisions of a specific natural gas plant consisting of 14 components which may be difficult to be applied for other kinds of systems.

Besides, recent advances in monotonically decomposing joint action-value functions allow to improve MADRL algorithms' scalability as well as training stability. Among the papers employing this technique, Weighted QMIX (WQMIX) (Rashid, Farquhar, Peng, & Whiteson, 2020) is one of the state-of-the-art algorithms, however, its performance for maintenance decision-making has not been investigated yet. Furthermore, WQMIX adopts the centralized training and decentralized execution paradigm which may cause slow learning in applications where agents can fully observe system states.

To address these issues, in this paper, we customize WQMIX to effectively optimize maintenance decisions of large-scale multi-component systems for the fully observable setting. In particular, separate agent networks are replaced by a single branching dueling network (branching network) (Tavakoli, Pardo, & Kormushev, 2018) to take advantage of the fully observable setting. The branching structure allows achieving a linear increase in the size of deep Q-networks's output layer to avoid the cure of dimentionality. Moreover, it also allows to create virtual communication channels between learning agents to facilitate decision-making processes as well as to avoid the use of recurrent neural networks in agent networks in the original paper that may slow down learning processes.

Our main contributions in this study are two folds. Firstly, we customize WQMIX algorithm specifically for the fully observable setting . Secondly, we conduct a comparison study to benchmark the performance of the customized algorithm, the branching dueling deep Q-learning (Tavakoli et al., 2018) and a threshold-based policy when they are used to optimize maintenance actions of large-scale systems.

The rest of the paper is organized as following. Section 2 is devoted to the general description of the maintained system. Maintenance operations and optimization problem statement formulation are described in section 3. The fully cooperative multi-agent setting for maintenance decision-making is depicted in section 4 and the detail of maintenance optimization process is presented in section 5. The numerical results are depicted and analyzed in section 6. The conclusions drawn from this work and some perspectives are presented in the last section.

2. System Description

We consider a series-parallel system being composed of N components which can be grouped into M subsystems. It is assumed that subsystem i contains H^i components of the same type i. As a result, $N = \sum_{i=1}^{M} H^i$.

A component of type i at a periodical inspection time t_k can be observed in any discrete health state, $s_k^i \in \{0, ..., m^i\}$, ranging from new to complete failure. Furthermore, it is also assumed that without any maintenance intervention, the state transition of a component of type i between two successive inspections obeys its inherent Markov probability transition matrix that has the following form:

$$\boldsymbol{P}^i = \begin{bmatrix} p_{00}^i & p_{01}^i & p_{02}^i & \cdots & p_{0m_i}^i \\ 0 & p_{11}^i & p_{12}^i & \cdots & p_{1m_i}^i \\ 0 & 0 & p_{22}^i & \cdots & p_{2m_i}^i \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \quad (1)$$

in which p_{uv}^i is a non-negative real number representing the degradation transition probability from state u to state v of a component of type i that satisfies: $\sum_{v=u}^{m^i} p_{uv}^i = 1, \forall u \in \{0, \dots, m^i\}$.

As an example, figure 1 illustrates a 13-component series system with four parallel subsystems. Specifically, component 1 is considered as the first subsystem. Component 2, 3 and 4 together form the next subsystem. The third one is composed of component 5, 6, 7 and 8. The last subsystem consists of the remaining components.

The degradation transition matrices of the four component

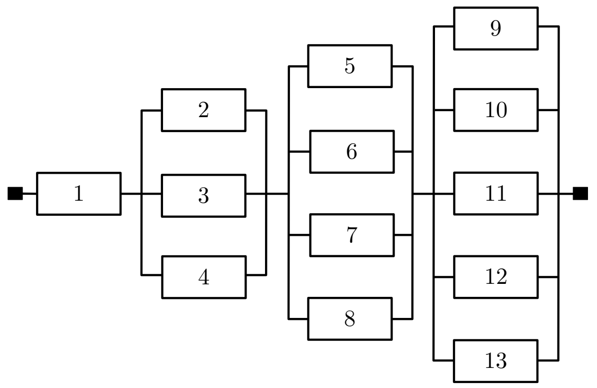


Figure 1. Reliability block diagram of the studied system

types are given as belows:

$$\boldsymbol{P}^1 = \begin{bmatrix} 0.60 & 0.30 & 0.05 & 0.05 \\ 0.00 & 0.60 & 0.30 & 0.10 \\ 0.00 & 0.00 & 0.60 & 0.40 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix} \tag{2}$$

$$\boldsymbol{P}^2 = \begin{bmatrix} 0.50 & 0.30 & 0.10 & 0.10 \\ 0.00 & 0.50 & 0.30 & 0.20 \\ 0.00 & 0.00 & 0.50 & 0.50 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix} \tag{3}$$

$$\boldsymbol{P}^3 = \begin{bmatrix} 0.65 & 0.25 & 0.05 & 0.05 \\ 0.00 & 0.65 & 0.25 & 0.10 \\ 0.00 & 0.00 & 0.65 & 0.35 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix} \tag{4}$$

$$\boldsymbol{P}^4 = \begin{bmatrix} 0.60 & 0.30 & 0.05 & 0.05 \\ 0.00 & 0.60 & 0.30 & 0.10 \\ 0.00 & 0.00 & 0.60 & 0.40 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix} \tag{5}$$

3. Maintenance Operations

Maintenance operations of the studied system are planned following both CM and PM strategy. A CM intervention replaces a failed component by a new one of the same type. PM actions could be either perfect or imperfect. While a perfect PM action completely restores a survival component to be as new, the imperfect one implies that state after maintenance of a component is somewhere between its state before maintenance and "as good as new" state. Moreover, it should be noted that maintenance actions can only be carried out after component states are revealed by inspections. As a result, when the failure of a component or a group of components occurs between two consecutive inspections, maintenance actions must wait until next scheduled inspection to be implemented.

The cost of maintaining a component individually consists of an inspection cost, a setup cost and a component-specific maintenance cost. The inspection cost is denoted as c^{ins}, which is necessarily paid even for survival components in order to reveal their current degradating states. We consider in this paper two levels of setup cost following (Wijnmalen & Hontelez, 1997), which are system setup cost, c^0, caused by, for example, transportation of spare parts or administrative handling, and component-type setup cost, $c^{t,i}$, originated from the requirement of specific tools or repairman skills. The component-specific maintenance cost is denoted as $c^{m,i}$ which depends on maintenance quality. Based on the above descriptions, the cost of separately maintaining a component i is computed as belows:

$$c^i = c^0 + c^{t,i} + c^{ins} + c^{m,i} \tag{6}$$

In practice, maintenance operations of multi-component systems usually benefits from shared setup costs when several components are grouped to maintain thanks to the positive economic dependency between them. For the studied system, the system setup cost can only be charged once if several components are maintained together. In the same manner, the component-type setup costs are charged once if a group of components of the same type are maintained simultaneously. As a result, the total maintenance cost at system level denoted as c can be calculated as follows:

$$c = \sum_{i=1}^{N} c^i - \mathbb{I}^0(N^0 - 1)c^0 - \sum_{i=1}^{M} \mathbb{I}^{m,i}(H^{m,i} - 1)c^{t,i} \tag{7}$$

in which N^0 is the number of maintained components; $H^{m,i}$ is the number of maintained components of subsystem i; $\mathbb{I}^0$ is the system maintenance indicator whose value is equal to one if there is at least one component being maintained or equal to zero otherwise; $\mathbb{I}^{m,i}$ is the maintenance indicator of type i whose value is equal to one if there is at least one component of type i being maintained or equal to zero otherwise. In addition to the maintenance cost, the downtime cost denoted as c^{dt} that is caused by the failure of a component or a group of components leading to the shutdown of the system should be considered. Our objective of maintenance decision-making optimization is to minimize to the long-run average cost rate.

4. Fully Cooperative Multi-agent Setting for Maintenance Decision-making

4.1. Agent-environment interaction

The maintenance optimization problem of the studied system is modeled as a fully cooperative multi-agent decision-making task with a group of N agents, $\mathcal{AG} = \left\{AG^i\right\}_{i=1}^{N}$, in which one agent controls maintenance decisions of one component and can fully observe system states. A component i has its own state space, $\mathcal{S}^i$, that help form the state space at system level, $\mathcal{S}^{joint} \equiv \mathcal{S}^1 \times \mathcal{S}^2 \times ... \times \mathcal{S}^N$. At any inspection time t_k, each agent $AG^i \in \mathcal{AG}$ observes the system's current state $\boldsymbol{s}_k \in \mathcal{S}^{joint}$ and then choose an action a_k^i from its own action space, $\mathcal{A}^i$, based on the observation and its own policy denoted as π^i. The actions chosen by individual agents form a joint action $\boldsymbol{a}_k \in \mathcal{A}^{joint} \equiv \mathcal{A}^1 \times \mathcal{A}^2 \times ... \times \mathcal{A}^N$. Once the chosen joint action is implemented, the system transitions to a state after maintenance $\bar{\boldsymbol{s}}_k$ and releases a numerical reward r_k shared by all agents. After that, it degrades naturally to a next state before maintenance $\boldsymbol{s}_{k+1}$ at inspection time t_{k+1} according to the transition matrices $\boldsymbol{P}^i$ ($i = \{1, ..., M\}$).

4.1.1. Environment element definition

The definition of system state, action and reward distribution mechanism within the context of this study is presented in the following paragraphs.

System state The state of a component i at inspection time t_k is its degradation level s_k^i. Hence, the system state at that time is a vector consisting all component states defined as $\boldsymbol{s}_k = \left[s_k^1, s_k^2, \ldots, s_k^N\right]^T$.

System action Similarly, the action at system level at inspection time t_k is a vector being composed of all component maintenance actions which is mathematically defined as: $\boldsymbol{a}_k = \left[a_k^1, a_k^2, \ldots, a_k^N\right]^T$ where a_k^i is the maintenance action of component i at that time. More specifically, there are three possible maintenance actions for each component which are encoded as belows:

$$a_k^i = \begin{cases} 0 & \text{leave component } i \text{ as it is} \\ 1 & \text{perform imperfect maintenance on component } i \\ 2 & \text{replace component } i \text{ by the one of the same type} \end{cases} \tag{8}$$

Imperfect maintenance implies that a component can be maintained to be in a better state which is some where between its current state and "as good as new". We employ the imperfect maintenance model in (Do & Bérenguer, 2012) where state after maintenance of a component can be obtained by sampling uniformly discretely from the interval from new state to its state before maintenance. The cost of maintaining a component i is computed as belows:

$$c_k^{m,i} = c_k^{r,i} . \left(\frac{s_k^i - \bar{s}_k^i}{s_k^i}\right)^\beta \tag{9}$$

in which $c_k^{r,i}$ is a constant representing the replacement cost of component i; s_k^i and $\bar{s}_k^i$ are respectively the state before and after maintenance of component i; β is a real positive number representing the components' imperfect maintenance characteristics.

Reward Maintenance optimization involves balancing the trade-off between maintenance frequency and downtime cost, which means that if maintenance is conducted too often, maintenance cost can be high or if maintenance operations are conducted less frequently, downtime cost is more prone to occurrence. To deal with this issue, the reward function used in this work is defined as the opposite of the total cost which is the sum of the maintenance cost at system level and the downtime cost as follows:

$$r_k = -c_k - \mathbb{I}_k c^{dt} \tag{10}$$

in which $\mathbb{I}_k$ is the system failure status indicator at time t_k whose value is equal to one if the system is in failed state at that time or is equal to zero otherwise; c^{dt} is a real constant representing downtime cost.

5. MADRL-Based Maintenance Optimization

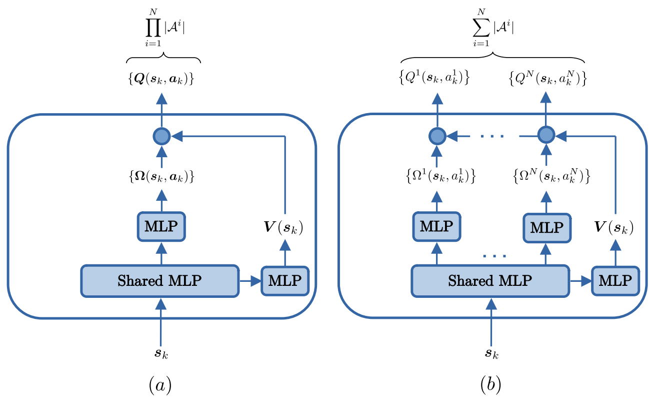
We first present BDQ algorithm to introduce the branching network in subsection 5.1 and then the customized WQMIX algorithm in subsection 5.2.

5.1. Branching dueling Q-learning (BDQ)

DRL has emerged recently as an effective framework for solving decision-making tasks with large state spaces by using deep neural networks to approximate action-value functions. This parameterized functional form allows to reduce the problem of determining values at each point in a Q-table to determining the number of weights of the corresponding network which is much less than the number of state-action pairs (Andriotis & Papakonstantinou, 2019). Moreover, the weight sharing in neural networks enables the generalization in the sense that updating weights for a single state-action pair affect the estimation of action values of other state-action pairs. Despite the success of DRL algorithms for applications with state space complexities, they may not be efficient for the ones with large action spaces due to the fact that the output layer of a deep Q-net work or a dueling deep-Q network consist of Q-values for each available actions. As a result, its size is equal to the size of action space (Andriotis & Papakonstantinou, 2019). For the system studied in this paper, the size of action space is equal to $\prod_{i=1}^{N} |\mathcal{A}^i|$ which grows exponentially in the number of components.

To tackle this problem, the BDQ algorithm in (Tavakoli et al., 2018) provides a special network structure, namely, branching dueling deep Q-network (or branching network for short), that allows the number of outputs of deep Q-networks to linearly increases with the number of components as illustrated in figure 2.

Specifically, at component level, each agent AG^i chooses a maintenance decision $a_k^i = \operatorname{argmax}_{a_k^i \in \mathcal{A}^i} Q^i(\boldsymbol{s}_k, a_k^i)$ based on its own action-value function which is computed based on its own advantage function, $\Omega^i(\boldsymbol{s}_k, a_k^i)$ and the state-value



$\prod_{i=1}^{N}|\mathcal{A}^i|$
$\{\boldsymbol{Q}(\boldsymbol{s}_k,\boldsymbol{a}_k)\}$
$\{\Omega(\boldsymbol{s}_k,\boldsymbol{a}_k)\}$
MLP
$\boldsymbol{V}(\boldsymbol{s}_k)$
Shared MLP
MLP
$\boldsymbol{s}_k$
(a)
$\sum_{i=1}^{N}|\mathcal{A}^i|$
$\{Q^1(\boldsymbol{s}_k,a_k^1)\}$
$\{Q^N(\boldsymbol{s}_k,a_k^N)\}$
$\{\Omega^1(\boldsymbol{s}_k,a_k^1)\}$
$\{\Omega^N(\boldsymbol{s}_k,a_k^N)\}$
MLP
MLP
$\boldsymbol{V}(\boldsymbol{s}_k)$
Shared MLP
MLP
$\boldsymbol{s}_k$
(b)

Figure 2. (a) Dueling Q-network structure. (b) Branching dueling Q-network structure.

function that is shared by all agents, $\boldsymbol{V}(\boldsymbol{s}_k)$, as belows:

$$Q^i(\boldsymbol{s}_k,a_k^i)=\boldsymbol{V}(\boldsymbol{s}_k)+\Omega^i(\boldsymbol{s}_k,a_k^i)-\frac{1}{|\mathcal{A}^i|}\sum_{a_k\in\mathcal{A}^i}\Omega^i(\boldsymbol{s}_k,a_k^i) \tag{11}$$

At system level, the agents cooperatively resolve to select a joint maintenance action $\boldsymbol{a}_k$ which is defined in previous section as a vector of all component maintenance actions according to the following equation:

$$\boldsymbol{a}_k=\begin{bmatrix}\underset{a_k^1\in\mathcal{A}^1}{\operatorname{argmax}}\,Q^1(\boldsymbol{s}_k,a_k^1)\\ \underset{a_k^2\in\mathcal{A}^2}{\operatorname{argmax}}\,Q^2(\boldsymbol{s}_k,a_k^2)\\ \dots\\ \underset{a_k^N\in\mathcal{A}^N}{\operatorname{argmax}}\,Q^N(\boldsymbol{s}_k,a_k^N)\end{bmatrix} \tag{12}$$

The loss of one transition sample, $(\boldsymbol{s}_k,\boldsymbol{a}_k,r_k,\boldsymbol{s}_{k+1})$, used to train the branching network is aggregated between all branches as belows:

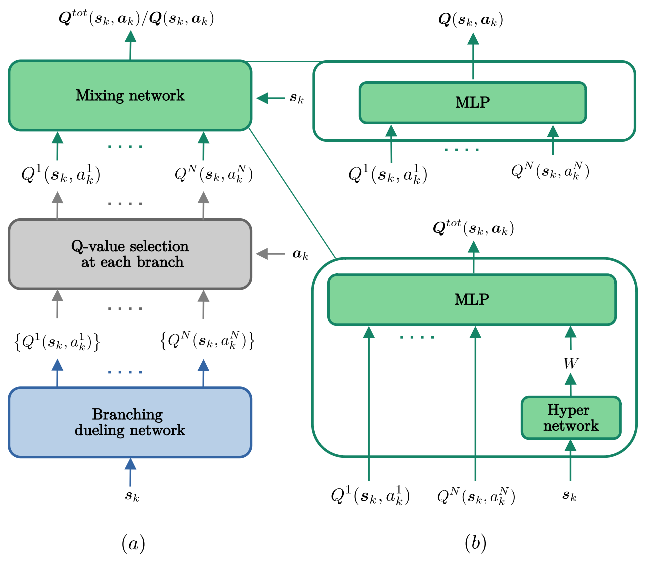
$$L=\frac{1}{N}\sum_{i=1}^{N}(Q^i(\boldsymbol{s}_k,a_k^i)-y)^2 \tag{13}$$

in which $y=r+\gamma\frac{1}{N}\sum_{i=1}^{N}Q^i_{target}(\boldsymbol{s}_{k+1},a_{k+1}^{i,*})$ is considered as the target which is shared between all branches where $a_{k+1}^{i,*}=\operatorname{argmax}_{a_{k+1}^i\in\mathcal{A}^i}Q^i(\boldsymbol{s}_{k+1},a_{k+1})$. It should be noted that Q^i_{target} is computed from a separate branching network called "target network" whose weights are periodically copied from the one used to calculate Q^i after every fixed number of training steps.

5.2. The customized WQMIX

Despite the advantage of the BDQ's branching network that allows to deal with the exponential increase in the number of outputs of deep Q-networks for high-dimensional systems, its training scheme is based on the idea of distributing temporal-difference errors across all branches, which is a heuristic approach and lacks of theoretical methodology. Indeed, BDQ does not guarantee one of the most important concepts of multi-agent systems which is the decision selection consistency between component and system level in the sense that learning agents cooperate with each other to choose a joint maintenance action $\boldsymbol{a}_k$ according to the system action-value function, $\boldsymbol{Q}(\boldsymbol{s}_k,\boldsymbol{a}_k)$, that should be consistent with actions chosen by local agents. The action selection consistency is mathematically expressed as:

$$\underset{\boldsymbol{a}_k\in\mathcal{A}}{\operatorname{argmax}}\,\boldsymbol{Q}(\boldsymbol{s}_k,\boldsymbol{a}_k)=\begin{bmatrix}\underset{a_k^1\in\mathcal{A}^1}{\operatorname{argmax}}\,Q^1(\boldsymbol{s}_k,a_k^1)\\ \underset{a_k^2\in\mathcal{A}^2}{\operatorname{argmax}}\,Q^2(\boldsymbol{s}_k,a_k^2)\\ \dots\\ \underset{a_k^N\in\mathcal{A}^N}{\operatorname{argmax}}\,Q^N(\boldsymbol{s}_k,a_k^N)\end{bmatrix} \tag{14}$$



$\boldsymbol{Q}^{tot}(\boldsymbol{s}_k,\boldsymbol{a}_k)/\boldsymbol{Q}(\boldsymbol{s}_k,\boldsymbol{a}_k)$
Mixing network
$\boldsymbol{s}_k$
$Q^1(\boldsymbol{s}_k,a_k^1)$
$Q^N(\boldsymbol{s}_k,a_k^N)$
Q-value selection at each branch
$\boldsymbol{a}_k$
$\{Q^1(\boldsymbol{s}_k,a_k^1)\}$
$\{Q^N(\boldsymbol{s}_k,a_k^N)\}$
Branching dueling network
$\boldsymbol{s}_k$
(a)
$\boldsymbol{Q}(\boldsymbol{s}_k,\boldsymbol{a}_k)$
MLP
$Q^1(\boldsymbol{s}_k,a_k^1)$
$Q^N(\boldsymbol{s}_k,a_k^N)$
$\boldsymbol{Q}^{tot}(\boldsymbol{s}_k,\boldsymbol{a}_k)$
MLP
W
Hyper network
$Q^1(\boldsymbol{s}_k,a_k^1)$
$Q^N(\boldsymbol{s}_k,a_k^N)$
$\boldsymbol{s}_k$
(b)

Figure 3. (a) The customized WQMIX architecture. (b) Mixing network structures.

Fortunately, the action selection consistency between component and system level can be achieved through the factorization method of VDN (Sunehag et al., 2017) which supposes that the system Q-function can be approximated by the sum of the per-component ones:

$$\boldsymbol{Q}(\boldsymbol{s}_k,\boldsymbol{a}_k)\approx\boldsymbol{Q}^{tot}(\boldsymbol{s}_k,\boldsymbol{a}_k)=\sum_{i=1}^{N}Q^i(\boldsymbol{s}_k,a_k^i) \tag{15}$$

QMIX (Rashid et al., 2018) generalizes the VDN's linear representation by assuming that $\boldsymbol{Q}^{tot}(\boldsymbol{s}_k,\boldsymbol{a}_k)$ is a monotonic continuous function of the per-agent Q-functions, in other words, $\partial\boldsymbol{Q}^{tot}(\boldsymbol{s}_k,\boldsymbol{a}_k)/\partial Q^i(\boldsymbol{s}_k,a_k^i)\geq 0,\forall i\in\{1,...,N\}$. This assumption can be realized by using a multi-layer perceptron (MLP) called mixing network that takes agents' own Q-values as inputs and outputting values of $\boldsymbol{Q}^{tot}(\boldsymbol{s}_k, \boldsymbol{a}_k)$, whose weights are generated by a hyper network (Ha, Dai, & Le, 2016) to assure their values greater than or equal to zero.

The monotonic factorization scheme restricts an agent choosing its own actions independent of the actions chosen by other agents which may lead to the finding of suboptimal polices for applications required strong cooperation efforts between learning agents as in the case of maintenance decision-making for multi-component systems. (Rashid et al., 2020) showed that this factorization limit originates from the equal weighting placed on each join action in the loss function used to update the joint Q-function, and proposed a weighting scheme to cope with this issue. Specifically, a MLP without any restriction to its weights is used to estimate system action values which is considered as baselines to put more attention on potential optimal joint actions in the loss function.

The network architectures used for computing $\boldsymbol{Q}(\boldsymbol{s}_k, \boldsymbol{a}_k)$ and $\boldsymbol{Q}^{tot}(\boldsymbol{s}_k, \boldsymbol{a}_k)$ is illustrate in figure 3. The losses of one transition sample, $(\boldsymbol{s}_k, \boldsymbol{a}_k, r_k, \boldsymbol{s}_{k+1})$, used to update the weights of these networks are computed as belows:

$$\begin{aligned} L^{\boldsymbol{Q}^{total}} &= w(\boldsymbol{s}_k, \boldsymbol{a}_k)(\boldsymbol{Q}^{total}(\boldsymbol{s}_k, \boldsymbol{a}_k) - y)^2 \\ L^{\boldsymbol{Q}} &= (\boldsymbol{Q}(\boldsymbol{s}_k, \boldsymbol{a}_k) - y)^2 \end{aligned} \tag{16}$$

where:

- $y = r + \gamma \boldsymbol{Q}_{target}(\boldsymbol{s}_{k+1}, \mathrm{argmax}_{\boldsymbol{a}_{k+1}} \boldsymbol{Q}^{tot}(\boldsymbol{s}_{k+1}, \boldsymbol{a}_{k+1}))$ is the fixed target with $\boldsymbol{Q}_{target}$ is computed from the target networks of $\boldsymbol{Q}$.
- $w(\boldsymbol{s}_k, \boldsymbol{a}_k)$ is the weight of joint action $\boldsymbol{a}_k$ whose value is equal to 1 if $\boldsymbol{Q}^{tot}(\boldsymbol{s}_k, \boldsymbol{a}_k) < y$ or equal to $\alpha \in (0, 1]$ otherwise.

6. NUMERICAL STUDIES

This section compares the performance of the customized WQMIX with BDQ and a threshold-based policy for maintenance optimization of the 13-component system depicted in figure 1.

6.1. System parameters

All cost parameters are given in arbitrary units (acu) which are presented in the following. The inspection cost c^{ins} and system setup cost c^0 are 5 and 30 (acu) respectively. The component setup cost of type i is $c^{t,i} \in \{25, 20, 15, 10\}$ for $i = 1, 2, 3, 4$. The replacement costs $c^{r,i}$ of four component types are 65, 60, 55, 50 (acu) respectively. The downtime cost constant is $c^{dt} = 1000$ (acu). Finally, the imperfect maintenance parameter β is set to 3.

6.2. Training descriptions

The branching network of WQMIX and BDQ takes component states as input, hence, its input layer's size is 13. The shared MLP consists of two layers of 128 hidden units and the advantage MLP for each branch and the MLP used for computing system value function are composed of a single layer of 64 and 128 hidden units respectively. The number of outputs in each branch is equal to the number of maintenance actions at component level which is 3. The mixing network of $\boldsymbol{Q}^{tot}$ consists of two hidden layers of 64 units, whose weights are generated by a two separate hyper-networks of 64 units. The mixing network of $\boldsymbol{Q}$ is a MLP of two hidden layers of 64 units.

It should be noted that due to the maintenance constrain described in section 3 that if a component is failed, it can only be replaced by a new one of the same type or be left as it is, we classify a component action, a_k^i, chosen at a given system state s_k^i as a wrong action if $s_k^i = m^i$ then $a_k^i = 1$, or as a feasible action, otherwise. In order to realize this constrain, an action mask is applied to filter out invalid actions. In particular, Q-value corresponding to wrong actions at each branch are forced to be $-\infty$ to guarantee that invalid actions cannot be chosen by DRL agents.

The two MADRL algorithms are trained through 2×10^6 steps. Learning rates are scheduled to decline from 10^{-3} to 0.25×10^{-3} during the first 300×10^3 training steps. Through training, exploration constant is annealed linearly from 1.0 to 0.05 over 500×10^3 training steps and kept as constant for the rest of the learning. A mini-batch size of 128 is used for uniformly sampling from relay buffers of 300×10^3 system transitions. The target update frequency is 20×10^3 steps. Latest policy networks after every 10^3 steps are employed to interact with a validation environment 5×10^3 times to compute corresponding cost rates.

The threshold-based maintenance policy used in this comparison study is originated from (Do & Bérenguer, 2012) which can be expressed by a vector $\boldsymbol{l} = \left[l^1, l^2, \ldots, l^N\right]$ where l^i is the preventive maintenance threshold of component i. The detail description of maintenance schedule of a component i is given in the following. If $s_k^i = m^i$, component i is in failed state. Thus, the "replacement" action is implemented immediately. If $l^i \leq s_k^i < m^i$, component i is still functioning but badly. Therefore, the "imperfect maintenance" action is carried out. If $s_k^i < l^i$, component i is functioning well. Accordingly, the "do nothing" action is chosen.

The optimal system-level preventive maintenance thresholds are obtained via genetic algorithm (GA). During optimization processes, each solution is evaluated using simulation results from 5 runs of 5×10^3 maintenance interventions. The GA optimizer is initialized with a population of 20 elements and a mutation rate of 0.1. The training converges after 25 iterations.

6.3. Simulation results

The simulation results are presented in figure 4, figure 5, table 1 and table 2. It can be noticed that the optimized cost rate of the threshold-based method is highest which is 326.53 (acu) with the corresponding preventive thresholds [1 2 2 2 2 2 2 2 2 2 2 2 2]. The cost rate obtained by BDQ is 265.75 (acu). In comparison with the two others, the customized WQMIX found the best cost rate which is 255.55 (acu).

Based on table 2, it can be seen that the optimization time of the threshold-based policy is shortest due to the fact that the search space of preventive maintenance thresholds is not too large for the studied system. The training time of the customized WQMIX is larger than the one of BDQ because of the extra neural networks used to compute $\boldsymbol{Q}$ and $\boldsymbol{Q}^{tot}$.

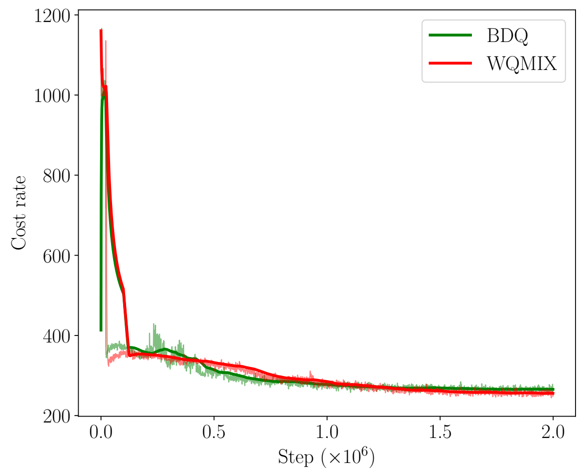


Figure 4. The evolution of cost rates during training

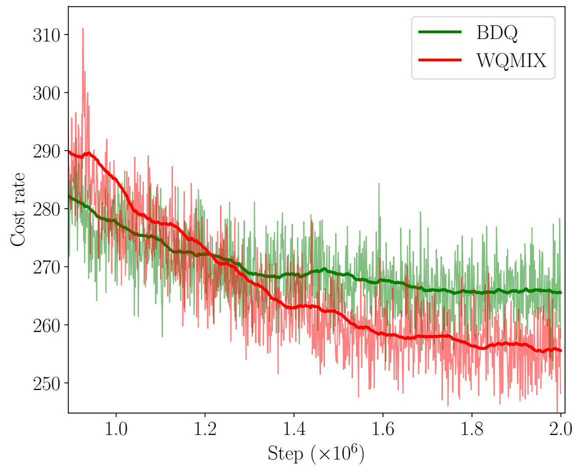


Figure 5. A closer look at the evolution of cost rates

Table 1. Cost rate summary (acu)

WQMIX	255.55
BDQ	265.75
Threshold-based policy	326.53

Table 2. Computing time summary (hours)

WQMIX	12.33
BDQ	8.68
Threshold-based policy	2.35

7. Conclusion

In this work, WQMIX algorithm is customized to effectively optimize maintenance decisions of large-scale systems in the case where system states can be fully observable. Particularly, separate agent networks are replaced by a single branching network to take advantage of the fully observable setting. The branching network reserves the ability of avoiding the cure of dimentionality as well as to facilitate decision-making processes. A comparative study is conducted on a 13-component system to examine the performance of the customized algorithm. The obtained results confirmed its effectiveness.

Our future work will focus on CBM modeling approaches for multi-component systems that can integrate multiple kinds of dependencies into maintenance models. Developing MADRL algorithms for maintenance decision optimization will be also considered.

Acknowledgment

This work is part of the AI-PROFICIENT project which has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 957391.

References

Ahmad, R., & Kamaruddin, S. (2012). An overview of time-based and condition-based maintenance in industrial application. *Computers & industrial engineering*, *63*(1), 135–149.

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

Do, P., & Bérenguer, C. (2012). Condition-based maintenance with imperfect preventive repairs for a deteriorating production system. *Quality and Reliability Engineering International*, *28*(6), 624–633.

Ha, D., Dai, A., & Le, Q. V. (2016). Hypernetworks. *ArXiv*.

Huang, J., Chang, Q., & Arinez, J. (2020). Deep reinforcement learning based preventive maintenance policy for serial production lines. *Expert Systems with Applications*, *160*, 113701.

Liu, Y., Chen, Y., & Jiang, T. (2019). Dynamic selective maintenance optimization for multi-state systems over a finite horizon: A deep reinforcement learning approach. *European Journal of Operational Research*, *283*(1), 166–181.

Quatrini, E., Costantino, F., Di Gravio, G., & Patriarca, R. (2020). Condition-based maintenance—an extensive literature review. *Machines*, *8*(2), 31.

Rashid, T., Farquhar, G., Peng, B., & Whiteson, S. (2020). Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., & Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International conference on machine learning* (pp. 4295–4304).

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., … others (2017). Value-decomposition networks for cooperative multi-agent learning. *ArXiv*.

Tavakoli, A., Pardo, F., & Kormushev, P. (2018). Action branching architectures for deep reinforcement learning. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 32).

Wang, H. (2002). A survey of maintenance policies of deteriorating systems. *European journal of operational research*, *139*(3), 469–489.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., & de Freitas, N. (2016). Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*.

Wijnmalen, D. J., & Hontelez, J. A. (1997). Coordinated condition-based repair strategies for components of a multi-component maintenance system with discounts. *European Journal of Operational Research*, *98*(1), 52–63.

Zhang, N., & Si, W. (2020). Deep reinforcement learning for condition-based maintenance planning of multi-component systems under dependent competing risks. *Reliability Engineering & System Safety*, *203*, 107094.

Zhou, Y., Li, B., & Lin, T. R. (2021). Maintenance optimisation of multicomponent systems using hierarchical coordinated reinforcement learning. *Reliability Engineering & System Safety*, 108078.