Contents lists available at ScienceDirect

Reliability Engineering and System Safety

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

Self-adaptive optimized maintenance of offshore wind turbines by intelligent Petri nets

Ali Saleh [a,b,*], Manuel Chiachío [a,b,**], Juan Fernández Salas [a,b], Athanasios Kolios [c]

[a] Andalusian Research Institute in Data Science and Computational Intelligence, University of Granada, 18071, Spain
[b] Dept. Structural Mechanics & Hydraulics Engineering, University of Granada, 18071, Granada, Spain
[c] Department of Wind & Energy Systems, Technical University of Denmark, RisøCampus Frederiksborgvej 399, DK-4000, Roskilde, Denmark

ARTICLE INFO

Keywords:
Petri net
Reinforcement learning
Q-learning
Offshore wind turbines
Condition-based maintenance

ABSTRACT

With the emerging monitoring technologies, condition-based maintenance is nowadays a reality for the wind energy industry. This is important to avoid unnecessary maintenance actions, which increase the operation and maintenance costs, along with the costs associated with downtime. However, condition-based maintenance requires a policy to transform system conditions into decision-making while considering monetary restrictions and energy productivity objectives. To address this challenge, an intelligent Petri net algorithm has been created and applied to model and optimize offshore wind turbines' operation and maintenance. The proposed method combines advanced Petri net modelling with Reinforcement Learning and is formulated in a general manner so it can be applied to optimize any Petri net model. The resulting methodology is applied to a case study considering the operation and maintenance of a wind turbine using operation and degradation data. The results show that the proposed method is capable to reach optimal condition-based maintenance policy considering maximum availability (equal to 99.4%) and minimal operational costs.

1. Introduction

Wind power is the fastest growing source of renewable energy in the industry in recent years [1]. Europe depends on wind energy to cover circa 15% of the electricity demand, indeed, only in 2019, the installed electricity capacity reached 205 GW [2]. Although most wind turbines are on land, a growing number of offshore wind turbines (OWT) are being installed, mostly due to greater availability of space, and reduced noise pollution and visual impact. Worldwide, the capacity of offshore wind farms has increased by more than 1500% between the years 2009 and 2021 reaching 55.6 GW [3].

Offshore wind farms are located in remote areas that are often subjected to harsh conditions, therefore, they require special equipment for maintenance (e.g., vessels), which significantly increases the operation and maintenance (O&M) costs [4]. Besides, maintenance operations typically require certain weather conditions, which increase the turbines' downtime and therefore, decrease the electricity production [5]. These high (O&M) costs impose a challenge to the offshore wind industry development; thus optimized maintenance plans have become a need to increase the competitiveness of offshore wind farms [6].

In recent years, a great effort has been made to help operators in scheduling optimal maintenance plans for OWT while considering influencing maintenance factors like site accessibility, quality of condition monitoring system (CMS), and weather conditions, to name but a few. Planning offshore interventions with the help of CMS can increase the availability of wind farms [7], and this can increase revenues. Yeter et al. studied the effect of inspection policy on the levelised cost of energy of OWT farm [8]. Nilsson and Bertling studied the effect of different maintenance strategies and CMS on running costs [9]. Marugán et al. considered the effect of incorrect data by using neural networks to detect false alarms aiming to increase system reliability [10]. Besnard et al. [11] created a model to optimize the maintenance as a function of environmental conditions, crew transfer vessels, crane ship availability, use of helicopters, and location of maintenance accommodation. Nurseda et al. studied the effect of wind speed and wind gust data on finding the optimal intervention time and execution order for maintenance tasks of corrective and periodic maintenance of an OWT farm [12]. Ghamlouch et al. used the theory of option to choose the best date for performing maintenance of a wind turbine based on the information provided by the CMS [13]. In [14], a method based on a time-sequential event-based Monte Carlo was used to optimize the maintenance while considering the effects of weather, price for vessels, workers' working hours, type of maintenance, failure rates, and electricity prices [14]. Similarly, Sahnoun et al. [15] developed a multi-criteria decision algorithm to account for the interactions of different

* Corresponding author.
** Andalusian Research Institute in Data Science and Computational Intelligence, University of Granada, 18071, Spain.
E-mail addresses: alisaleh@ugr.es (A. Saleh), mchiachio@ugr.es (M. Chiachío).

https://doi.org/10.1016/j.ress.2022.109013
Received 6 September 2022; Received in revised form 17 November 2022; Accepted 26 November 2022
Available online 5 December 2022

parts when optimizing the maintenance strategy, with consideration of water depth, distance from shore, wind quality, failure modes, accessibility, turbines quantity and rating, and availability of spare parts, workers, cranes and boats. Research conducted in this area also focused on reducing the O&M costs for OWT using a variety of multi-criteria optimization schemes [16–19]. However, those maintenance modelling and optimization techniques do not have learning capabilities, and that limits the capacity to autonomously adapt the resulting maintenance schedule to the changing nature of the influencing conditions.

The aforementioned approaches are based on a number of mathematical and computational techniques for system-level maintenance modelling, including discrete event simulation [20,21], Markov and semi-Markov models [22,23], agent-based modelling [24,25], soft-computing techniques like genetic algorithms and neural networks [26–28], and Petri nets [29–31], to name but the most relevant ones. Among them, Petri nets (PNs) are typically regarded as powerful modelling tools for maintenance and inspection modelling due to their ability to account for resource availability, concurrency, and synchronization, which are common aspects that underline the majority of the asset management models, along with their adequacy for dealing with highly multidimensional and heterogeneous input variables [32,33]. The basic concepts relative to the theory of PNs are summarized in [34], and a tutorial for engineering applications is given in [35]. However, the existing PN formalisms do not provide direct means to efficiently consider knowledge learning so as to make them self-adaptive to the variation of the environment.

In the literature, a number of PN variants have been presented to enhance the PN modelling capabilities with improved rules of learning, like fuzzy Petri nets (FPN) [36,37] and Possibilistic Petri nets [38], where the learning is limited to adjust fuzzy production rules using some soft-computing technique, and Plausible Petri nets [39], which are based on Bayesian learning. The latter has been demonstrated to be an effective tool for making PNs self-adaptive to varying conditions [40], however, the learning is limited to low dimensional and homogeneous variables. Other approaches encountered in the literature have enabled some sort of learning for PNs although they are domain or purpose-specific [41,42]. Notwithstanding these achievements, none of the PN variants developed is well-suited for embedding intelligence into the PN formulation to autonomously react to changes in the influencing environment by adaptive learning.

In this context, this paper proposes a new variant within the PN paradigm, named intelligent Petri net (*i*PN), obtained by combining the Reinforcement Learning (RL) technique with the principles of PNs. Reinforcement Learning is a computational method used for machines to learn from interaction with the environment by mimicking the behaviour of organisms [43]. The learner has to discover possible choices rather than being told which action is the best for each situation. A key feature of RL is that it considers the whole problem of an agent interacting with an uncertain environment in contrast to other approaches, like those cited above using multi-criteria optimization, which deals with the optimization of sub-problems instead of focusing on the final goal.

RL has demonstrated efficiency as a tool for scheduling asset management problems. Pinciroli et al. formulated the O&M optimization of renewable energy systems as a sequential decision problem that is solved through Deep Reinforcement Learning (DRL) [44]. Fan et al. optimized gas supply reliability by minimizing gas shortage risk and reducing maintenance costs through DRL [45]. The maintenance planning problem for aircraft was considered by Lee et al. using DRL [46], and by Mattila et al. using a RL method for semi-Markov processes [47]. The maintenance decisions of multi-state component systems were considered in different ways by Yang et al. using RL [48], and by Nguyen et al. using multi-agent deep Reinforcement learning (MADRL) [49]. Mohammadi et al. optimized the maintenance planning and renewal of a rail using Double Deep Q-Network (DDQN) [50]. Yang et al. optimized the condition-based maintenance (CBM) strategy for redundant systems using Reinforcement learning [51]. Kuhnle et al. optimized the working schedule and opportunistic maintenance in a system of parallel machines [52]. In the wind sector, the RL has shown its capabilities in different applications, including planning vessel transfers to offshore wind turbines in a cost-effective manner [53], improving the efficiency of wind turbine pitch controller [54,55] and creating an effective preventive maintenance action sequence [56].

There are some efforts that can be encountered in the literature about using RL with PNs models for system-level intelligent modelling. These approaches are mostly focused on robotic applications [57,58], and also for intelligent manufacturing modelling [59,60]. However, the RL in these approaches is not implicitly embedded with the PN model in the mentioned studies, which makes the methodology case specific.

In this work, a novel *i*PNs which combines RL and PN in a generic manner is proposed. This methodology enables the upgrade of any PN model from ordinary to intelligent with minimal change in the PN architecture. To formulate the *i*PNs, a new PN syntax element, named *action group*, is defined for each group of transitions that describes conflicting choices. By these means, the *i*PN can find and create state–action pairs automatically, evaluate the values of these pairs, and find the best choice in each action group as a function of the state of the environment. Among the RL paradigms, the Q-learning method, which resembles a Markov Decision Process (MDP), is adopted in this study [43,61]. To release the MDP assumption, a method is proposed to allow taking a varying number of simultaneous actions by considering the joint action similar to what is done in multi-agent Reinforcement learning (MARL). In addition, the method has been enriched with a new technique to apply the Q-learning equation when the termination of an episode occurs due to time constraints. The proposed methodology has been used to improve the reliability of an OWT by evaluating and optimizing its O&M. The proposed techniques to consider simultaneous actions and time termination become essential to decide regarding the repair of multiple components and to consider the end of the problem according to the lifespan of the wind turbine. The OWT case study has been addressed while considering several influencing O&M factors, including site accessibility, probability of failures, electricity prices, running costs, components reliability, reliability and quality of CMS, weather conditions, and lifespan of a wind turbine. As a result, the proposed *i*PN model has been able to autonomously find an optimal CBM policy by self-adaptation, which drastically reduces the O&M costs and increases the availability to more than 99%.

Apart from formulating the concept of *i*PNs and applying them for self-adaptive O&M, this manuscript also:

1. formally integrates the *i*PNs formulation for high-level Petri nets;
2. provides a pseudocode for the implementation of *i*PNs;
3. shows how the *i*PNs can be used to maximize the reliability and availability of complex engineering systems by finding an optimal O&M policy;
4. proposes a novel and holistic O&M PN-based model of an OWT with consideration of multiple interacting components, including the CMS;
5. provides a detailed discussion, in light of the results, about how the proposed methodology compares to those obtained using periodic and corrective maintenance schemes.

The rest of the paper is organized as follows. Section 2 describes the foundations of Q-learning and Petri nets, along with the proposed formulation needed to equip a PN with Q-learning. In Section 3 the pseudocode implementation of the proposed *i*PN algorithm is given along with other practical implementation aspects within the O&M modelling. Section 4 presents the operation and maintenance intelligent PN model created for an OWT. Section 5 presents the results of the OWT case, whereas the discussion of the results is given in Section 6. Finally, Section 7 provides concluding remarks.

2. Methodology

This section provides the methodological background required to formulate and applied the intelligent PN algorithm proposed in this work.

2.1. Reinforcement learning (Q-learning)

The RL is a machine learning method where the training process is based on rewarding and punishing an *agent* based on its behaviour within an *environment*, using a value function, a policy, and, optionally, a model of the environment. Through a Markov decision process (MDP), the agent should be able to sense a fully observable environment, take decisions related to the states of the environment, and learn from its own experience. In the MDP, the agent receives a representation of the environment, referred to as *state* $S_t \in \mathcal{S}$, at each time step t, where $\mathcal{S}$ is a set of possible states. Accordingly, an *action* A_t is taken within a set of possible actions named as $\mathcal{A}(s)$, that results in a reward $R_{t+1} \in \mathbb{R}$, whereupon a change to state S_{t+1} is triggered at time step $t+1$. The pair reward and state $\{R_{t+1}, S_{t+1}\}$ follows the Markov property, i.e., they are only dependent on the preceding state S_t and action A_t. Thus, to model the state evolution of the MDP, a probability distribution is defined as $p(s', r|s, a)$, representing the plausibility of obtaining the state $S_{t+1} = s'$ and its corresponding reward $R_{t+1} = r$ by taking an action $A_t = a$ at a state $S_t = s$.[1]

Starting from any time step t, the MDP results in a sequence of triads state, action, reward, as: $\{S_t, A_t, R_{t+1}\}, \{S_{t+1}, A_{t+1}, R_{t+2}\}, \dots, \{S_T\}$, until a termination time T is reached. In RL, the agent tends to take an action A_t that increases the expected return at time t, referred to as G_t, which is defined as the cumulative summation of rewards coming after that action, as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{1}$$

where $\gamma \in [0,1]$ is a discount rate parameter to prevent $G_t \to \infty$ when $T \to \infty$ (known as a *continuous task problems*). By the contrary, in *episodic task problems*, the terminating time step T is a finite number, thus G_t can be calculated by choosing $\gamma = 1$. In any case, the expected return G_t calculated after being at state S_t indicates how good it was to be at that state in terms of current and future rewards. However, note that G_t is not constant and it can vary because of the different agent's later actions $(A_{t+1}, A_{t+2}, \dots)$ that can be adopted and also because of the stochastic nature of the sequence of states through the referred probability distribution $p(s', r|s, a)$. Thus, a *value function* $v_\pi(s)$ takes any state s as an argument and returns its value, which is defined as the expectation of the returns following a policy π, as follows:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s], \tag{2}$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s], \forall s \in \mathcal{S} \tag{3}$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v_\pi(s')], \tag{4}$$

where $\pi(a|s)$ represents the policy which gives the probability for choosing each of the possible actions $a \in \mathcal{A}(s)$ given a state $s \in \mathcal{S}$. The resulting equation (Eq. (4)) is called the *Bellman equation* and is used to evaluate $v_\pi(s)$. This step is known as the *policy evaluation* and is followed by the *policy improvement* step to form the bases of the dynamic programming (DP). Policy improvement can be done by favouring actions that lead to the states with the highest values. However, different policies result in different sequences of rewards which affect the calculation of the expected returns and the values. Thus, the policy evaluation and improvement steps should be iterated until reaching an optimal policy. Notwithstanding, getting a model of the environment, given by $p(s', r|s, a)$, is difficult and can be unfeasible in some cases which makes the use of DP methods limited. Thus, model-free RL by temporal-difference learning (TDL) is widely used due to their simplicity and minimal amount of computation [43]. This method estimates the value function of state–action pairs, $q_\pi(s, a)$, instead of $v_\pi(s)$ to compare different actions at the same state and take a decision accordingly. By the bootstrapping effect, TDL makes use of the value of the successor state to update that of the current state, thus allowing the updating of the values online while interacting with the environment without the need to wait until the end of the rewards sequence. One of the earliest and most famous TDL methods is the *Q-learning* [61], whose update equation is given as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{5}$$

where $Q(S_t, A_t)$ is the value of the state–action pair (S_t, A_t) and is known as the Q-value, and α is a *learning rate* parameter used to keep the effect of old returns. The Q-values of all state–action pairs approximate the optimal value function $q^*_\pi(s, a)$ regardless of the followed policy [43]. The policy can be improved as a function of Q-values by following the *greedy strategy* [43], as follows:

$$\pi = \arg\max_a Q(S, a) \tag{6}$$

Note that the *greedy strategy* lacks exploration capability since it does not allow trying actions other than the ones with the highest Q-values. At the same time, Q-values in Eq. (5) are not converged at the beginning of the learning process, which may drive the agent to stuck on choosing non-optimal actions after being deluded that their values are better than the non tried ones. An alternative is to use the *ε-greedy* method to make a balance between exploration and exploitation by keeping a probability of choosing random actions alive and equals to ε, as follows:

$$A_t = \begin{cases} \arg\max_a Q(S, a), & \text{with probability } (1\text{-}\varepsilon) \\ A \in \mathcal{A}(s), & \text{with probability } \varepsilon \end{cases} \tag{7}$$

where A is a random action within the set $\mathcal{A}(s)$. This method guarantees that each action will be selected infinitely if the state is visited infinite times, ensuring that the exploration of all actions continues as long as the optimization process is running. This also ensures the convergence of the Q-values of the possible state–action pairs while maximizing the rewards. In a stationary environment, an agent can start reducing exploration when gaining more experience. Thus, ε can be chosen to be a variable that decreases gradually as the learning process proceeds, and this is known as the *decaying-ε-greedy* method [43]. In this paper, the *Q-learning* with *decaying-ε-greedy* strategy is followed to make the PN model intelligent, as will be indicated next in Section 3.

2.2. Industrial modelling with Petri nets

Petri nets (PN) are directed bipartite graphs, which use two type of nodes: *transitions*, represented using small rectangles, and *places*, symbolized using circles, which are visited by *tokens*, the moving units of the PN that describe the states of a PN according to their distribution over the places. This distribution of tokens at a specific execution time is referred to as *marking*. Transitions are the units responsible for the dynamics of a PN by consuming and producing tokens in their connected places. The arcs connecting transitions and places have positive-integer labels that represent the *weights* and indicate the number of parallel arcs (1 by default) connecting the nodes. The PN symbols used in this paper are schematically indicated in Fig. 1.

Mathematically, a PN is defined as a tuple [34]:

$$\mathfrak{P} \triangleq \left\langle \mathbf{P}, \mathbf{T}, \mathbf{F}, \mathbf{M}_0, \mathbf{W} \right\rangle \tag{8}$$

[1] R_t, S_t, and A_t represent specific reward, state, and action at time step t while r, s, and a are used to refer to the reward, state, or action when passed as arguments (dummy variables) to functions, respectively.

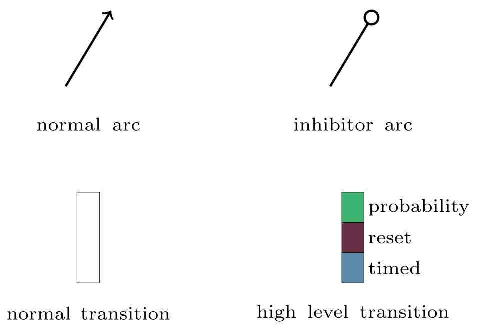


Fig. 1. Specification of the different symbols used in the Petri net (PN) models.

where $\mathbf{P} = \{p_1, p_2, \dots, p_{n_p}\}$ and $\mathbf{T} = \{t_1, t_2, \dots, t_{n_t}\}$ denote finite sets of n_p places and n_t transitions of the PN respectively, and $\mathbf{M}_0 \in \mathbb{N}^{n_p}$ is the initial marking. The connections between nodes are expressed through the set of edges $\mathbf{F} \subseteq (\mathbf{P}\times\mathbf{T}) \cup (\mathbf{T}\times\mathbf{P})$, so that $\mathbf{F}$ contains ordered pairs of nodes. Each edge (also referred to as arc) has assigned a weight defined in the set of weights $\mathbf{W}$.

The following notation will also be considered:

${}^{\bullet}\mathbf{P}_t$ is the set of input places of transition t, also referred to as the *pre-set* of t;

$\mathbf{P}_t^{\bullet}$ is the set of output places of transition t, also referred to as the *post-set* of t;

The rule controlling the consumption and the production of tokens by the transitions is called the *firing rule*, which states that each pre-set place of a transition has to be marked with a number of tokens equal to or greater than those indicated by the arc weight connecting that place to the transition, for the transition to be fired. Each time a transition fires, the marking and the state of the system change. The dynamics of a PN are formulated mathematically through the state equation defined by:

$$\mathbf{M}_{k+1} = \mathbf{M}_k + \mathbf{A}^T\mathbf{u}_k \tag{9}$$

where $\mathbf{M}_k$ is the marking vector at time step k and $\mathbf{u}_k = (u_{1,k}, u_{2,k}, \dots, u_{n_t,k})^T$ is the *firing vector*. $\mathbf{A}$ is a $n_t \times n_p$ matrix typically referred to as the *incidence matrix*, which can be obtained as the result of subtracting the *backward incidence matrix* $\mathbf{A}^- = \left[a_{ij}^-\right]$ from the *forward incidence matrix* $\mathbf{A}^+ = \left[a_{ij}^+\right]$, where the terms a_{ij}^- and a_{ij}^+ coincide with the weights w_{ij}^- and w_{ij}^+ of the arcs of the transition t_i from its pre-set and post-set places p_j, respectively.

In practical applications dealing with the modelling of dynamical systems, transitions are typically assigned time delays, given by τ, and these PNs are called Timed Petri Nets (TPN). The value of τ can be deterministic or given by a probability density function, thus the PN is referred to as stochastic Petri Net (SPN). Once the firing rule is satisfied in a TPN or a SPN, the transition is said to be enabled, however, it will not fire until the delay time passes while maintaining the enabled state of the transition.

Remark 1. Maintenance models at the system level usually require the use of PN variants known as high-level Petri nets (HLPN) [62], which allow the use of logic flow in a wider manner as well as considering high complexities. In these variants, the basic PN formalism is extended by incorporating logic and mathematical predicates within the PN structure or using malleable definitions of arc types, and tokens, along with transition firing rules.

In this paper, *inhibitor arcs* [34], *reset arcs* [63], and *probability transitions* [33] variants of HLPN are used (shown in Fig. 1), whose behaviour is described next. Pre-set places connected to a transition with inhibitor arcs (represented by the circular ending) prevent the enabling of the transition when they are marked. Post-set places connected to a transition through reset arcs (represented by the filled circular ending) return to their initial marking after that transition is fired. If a transition is probabilistic, the production of tokens in its post-set places is done with a specified probability, such that each time the transition fires, tokens are produced only through one of the post-set arcs that are chosen according to the probability [33].

2.3. Intelligent Petri net modelling by reinforcement learning

In addition to the high level variants described above, a finite set $\mathbf{G} = \{g_1, g_2 \dots, g_{n_g}\}$, named *action groups*, is introduced to the PN tuple to incorporate RL and upgrade it to an intelligent PN (*i*PN) model, through the following execution rules:

- each action group, g_i, is composed of a finite set of transitions, $\{t_{i,1}, \dots, t_{i,\ell}, \dots, t_{i,n_i}\}$, that represent decisions within a RL environment;
- transitions of the same action group should be able to satisfy the enabling conditions at the same PN state;
- an action group can be either enabled or disabled. If a transition $t_{i,\ell} \in g_i$ satisfies the PN enabling conditions by the firing rule:
  - the transition is not directly enabled;
  - its action group will become enabled;
  - if the action group was not enabled previously, a transition, that might be different from $t_{i,\ell}$, will be chosen by RL to be enabled using the RL policy and based on the RL state;
  - if $t_{i,\ell}$ is a timed transition, its time delay might give raise the action group to be enabled again. Thus, to prevent the action group from choosing another transition before the first one is finished, the action group should be kept enabled until the firing of the chosen transition.

In this approach, it is important to distinguish between the RL states and the *i*PN states. The RL states may include part or all of the markings (PN states), plus some additional information, and they should always provide the information the agent needs to make decisions. In the formulation presented in this paper, the modeller defines the variables needed by the agent when taking a decision through the adoption of an action. Thus, these variables form the environment, whereas the values of these variables constitute the RL states.

Fig. 2 shows an illustrative example of a high-level *i*PN used to model the maintenance of a generic component subject to degradation. The places p_1, p_2, p_3, and p_4 represent the normal, degraded, critical and failed states of the component. Transitions t_1, t_2, and t_4 represent the degradation steps of the component, whereas transitions t_3, t_5, and t_6 model the minor and major repairs, along with replacement of the component, respectively. Observe that transition t_2 is a probability transition with delayed firing, with probability π_{23} to change from p_2 to p_3, and π_{24} to change from p_2 to p_4, where $\pi_{23} + \pi_{24} = 1$. Note also that t_2 and t_3 are conflict transitions, which form the action

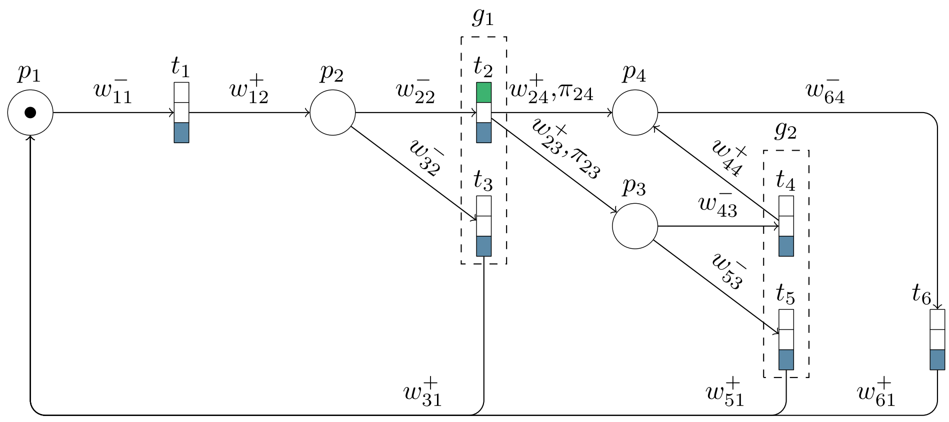


Fig. 2. High Level *i*PN of four places and six transitions used as illustrative example along with indication of its initial marking. Two action groups, g_1 and g_2, are used to embed the RL methods within the PN rules.

group g_1. The decision taken in g_1 determines if the component, once degraded (i.e., p_2 marked) will continue degrading, or a repair action is undertaken to recover the normal state. The same applies to action group g_2 when the component is in a critical state.

Two kinds of simulations were done for the PN shown in Fig. 2 to comparatively assess the performance of the *i*PN. The first simulation was carried out with no consideration of the proposed method, namely, the model is considered just as a HLPN where the action groups were removed and the choice of transitions was done in a random manner, as usual. The Monte Carlo simulation was employed to simulate the choice of the sequence of actions that produces the *total rewards per episode* among all runs. In the second simulation, the proposed *i*PN methodology was used. The parameters assigned to the PN in both simulations were chosen in an arbitrary way to keep them simple, namely: a delay equal to 1 time step (referred to as t_{stp}) for all transitions; reward equal to $10+0.2{\cdot}t_{stp}$, $30+0.1{\cdot}t_{stp}$, $40-0.3{\cdot}t_{stp}$, and 20, for transitions t_2, t_3, t_4, and t_5, respectively; probabilities π_{24} and π_{23} equal to 0 and 1, respectively; and a life span equals to 100 t_{stp}. The results of both simulations are shown in Fig. 3a and b, for the case of using an ordinary HLPN and the proposed *i*PN methodology, respectively. Observe that the HLPN simulated with Monte Carlo was not able to deal with any episode with total rewards greater than 3400 whilst the *i*PN was able to reach an average of 4000 by the end of the learning process.

Note as well that, in the considered case, action needs to be taken at least every two time steps resulting in at least 100 actions per episode of 200 time steps. The probability of choosing an optimal action at each step is 0.5 since only two actions are available at each step. Then, the probability of having 100 consecutive optimal actions is $0.5^{100} = 7.8 \cdot 10^{-31}$, which makes it an impossible task using random search optimization techniques. The bootstrapping method used in Q-learning is equivalent to dividing the problem into small steps optimization problems, which makes finding the optimal policy much faster and easier.

3. The *i*PN algorithm

This section provides an algorithmic description of a high-level *i*PN model by using the Q-learning method in a generic manner. A pseudocode implementation of the proposed *i*PN is given as Algorithm 1. Fig. 4 provides a flowchart to better clarify how the algorithm works. Note that an outer loop exists over the episodes, and each episode contains a loop that starts with a time equal to zero and ends when the time exceeds the *end time*, or when no more transitions are enabled. Observe also that each state of an episode passes three sub-algorithms that manage the enabling of the transitions, the action choice, and the firing of the transitions. After the marking and time conditions are satisfied, the transitions are fired, whereupon a change in the overall state of the system occurs. The algorithm also covers specific treatment of practical implementation aspects like the case of episodes with time-terminating states, and the case of actions with multiple decisions. These aspects are explained below in Sections 3.1 and 3.2, respectively. Also, indications about scaling the algorithm parameters are given next, in Section 3.3.

3.1. Treatment of episodes with time terminating states

In many practical applications, like the one about OWT presented in the case study below, the Q-learning problem can be considered as an episodic task with the termination occurring after the lifespan ends. In Q-learning of episodic tasks, the Q-value of the terminating state should always be 0, otherwise, the Q-values will diverge to ∞ when using a discount rate of 1 [43]. Accordingly, the terminating state is not an actual state (i.e., nothing in the environment changes when the termination happens), and any of the states has a probability of being a pre-terminating state. Each time the termination occurs, the last passed RL state is actually the pre-terminating and not the terminating state, and its Q-value should be updated by assuming that there is a terminating state with a Q-value equal to zero. Thus, for the case of episodes terminated due to a time-based condition, the Q-value update equation of the last passed RL state should be modified as follows:

$$\begin{aligned} Q(S_t,A_t) &\leftarrow Q(S_t,A_t)+\alpha[R_{t+1}+\gamma \cancelto{0}{\max_a Q(S_{t+1},a)}-Q(S_t,A_t)] \\ &= Q(S_t,A_t)+\alpha[R_{t+1}-Q(S_t,A_t)] \end{aligned} \tag{10}$$

Note that this aspect has been considered in line 51 of Algorithm 1.

3.2. Consideration of simultaneous actions

Again, in practical applications, a state with multiple actions to be taken simultaneously can occur, and each cannot be considered as a separate state–action pair when using Q-learning. This is because Q-learning is based on the MDP assumption, which states that the current state is a function of the previous state–action pair. Indeed, if multiple actions were taken sequentially given a state, each one would have a separate Q-value that would be updated as a function of another Q-value, that is in reality not preceding it. The simultaneous actions at the same state should be taken in parallel and this can be done through MARL [64]. However, the number of actions at each state can be variable, which requires a variable number of agents, so the ordinary MARL techniques may not be useful for the *i*PN methodology. For this, new definitions were made to incorporate the RL method with simultaneous actions:

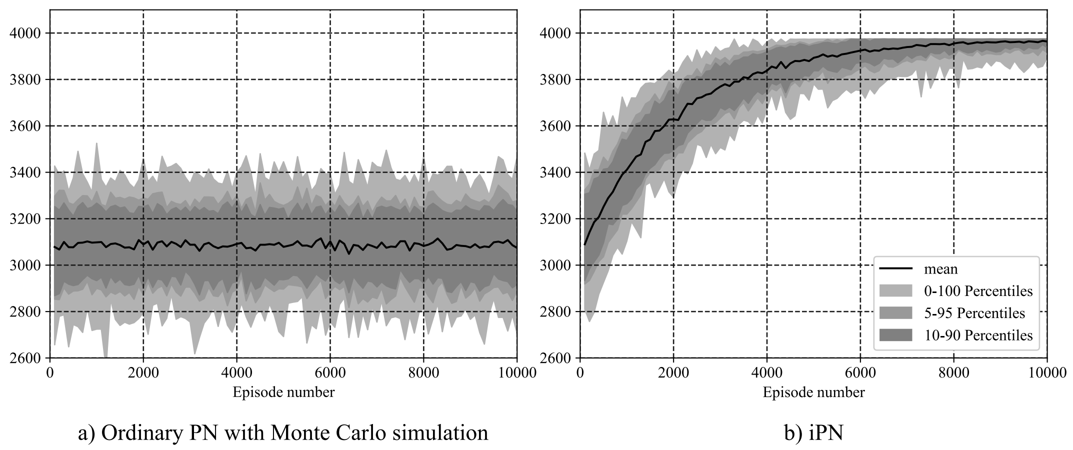
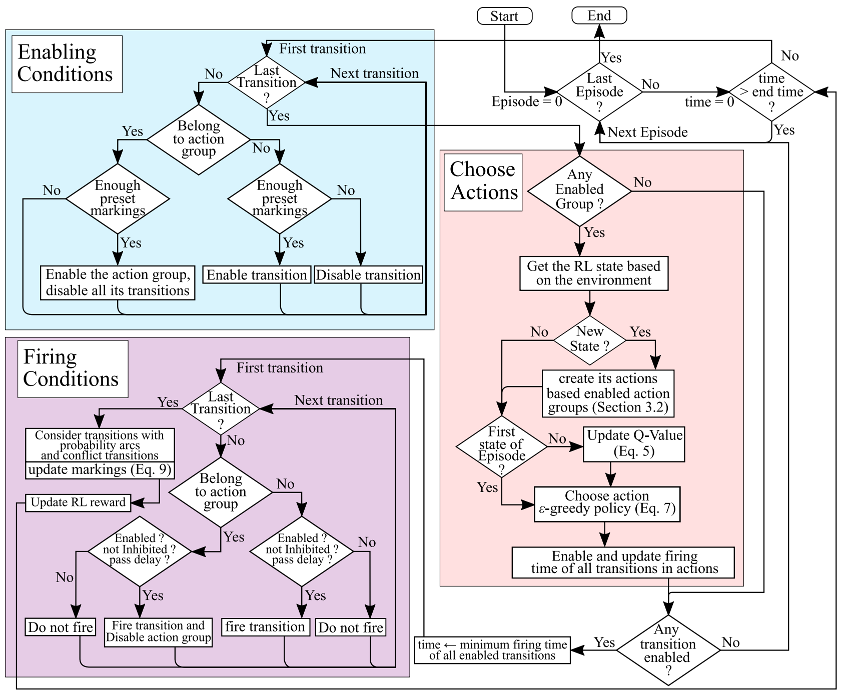


Fig. 3. Comparison of the *i*PN algorithm with the ordinary PN with Monte Carlo simulation for finding the optimal strategy that maximizes the total rewards per episode.



Enabling Conditions
Firing Conditions
Choose Actions
Start
End
Last Episode ?
time > end time ?
Episode = 0
time = 0
Next Episode
First transition
Last Transition ?
Next transition
Belong to action group
Enough preset markings
Enable the action group, disable all its transitions
Enable transition
Disable transition
Any Enabled Group ?
Get the RL state based on the environment
New State ?
create its actions based enabled action groups (Section 3.2)
First state of Episode ?
Update Q-Value (Eq. 5)
Choose action ε-greedy policy (Eq. 7)
Enable and update firing time of all transitions in actions
Any transition enabled ?
time ← minimum firing time of all enabled transitions
Consider transitions with probability arcs and conflict transitions
update markings (Eq. 9)
Update RL reward
Enabled ? not Inhibited ? pass delay ?
Do not fire
Fire transition and Disable action group
fire transition
Yes
No

Fig. 4. Flowchart of the proposed high level intelligent Petri net (*i*PN) methodology by Q-learning.

- the act of selecting a transition from an action group is defined as a decision rather than an action;
- an action group that is enabled and waiting for the selection of transition is called a pending action group. At RL state S, each pending action group will have a set of available decisions, $\mathcal{D}(S)$;
- at RL state S, the set of available actions, $\mathcal{A}(S)$, will be equal to the Cartesian product of all available decision sets: $\mathcal{A} = \mathcal{D}_1 \times \mathcal{D}_2 \ldots \times \mathcal{D}_n$, where n is the number of pending action groups;
- there will be no local Q-Values for each decision, but a global Q-Value linked to the state and the joint action and updated based on the sum of the rewards that come after the joint action. Thus, the single-agent RL method will be applied, although Q-Value will be related to the joint action instead of one decision;
- the policy selects a joint action that represents a combination of simultaneous decisions but does not select each decision separately.

Considering the joint action rather than separate actions ensures that the MDP assumption is obeyed since the method will become in the category of fully cooperative MARL methods with centralized controller [65], which guarantees the generality of the method in the case of simultaneous decisions.

3.3. Scaling of the Q-learning parameters

As indicated in line 2 of Algorithm 1, three parameters are used to scale the Q-learning method within the *i*PNs, namely, the discount rate γ, the learning rate α, and exploration rate ε. Section 2.1 already indicated that in episodic task problems (like the OWT case study considered in this work, and shown next), the discount rate can be assigned a value $\gamma = 1$.

Regarding the learning rate parameter, α, it is proposed to decay gradually throughout the learning process using Eq. (11). Note that this equation is composed of two parts, one for the first Q-value updates, while the other is for the rest of the updates, where the gradual decay actually applies. The referred equation is mathematically described as follows:

Algorithm 1: High level intelligent Petri net (*i*PN) by Q-learning

```
1  Definitions: $u$ is the firing state, $w$ is the enabling state, $x$ is the
     old enabling state, $\tau$ is the time delay, $\tilde{\tau}$ is the firing time, $c$ is
     the pending state of the action group (represent if transition
     was chosen after the action group was enabled), $\mathcal{D}(S)$ and
     $\mathcal{A}(S)$ are decisions and actions sets of state $S$
2  Parameters: Learning rate $\alpha \in (0,1]$; discount rate $\gamma \in (0,1]$;
     exploration rate $\varepsilon \in (0,1]$; end time of the simulation,
     $ET \in \mathbb{R}^+$.
3  Empty set of RL states $\mathcal{S} = \{\}$
4  Sub-Algorithm1: Enabling Conditions:
5    for All $t$ in T do
6      if $t \in g \backslash g \in$ G then // transition belong to any
         action group
7        if $w_g = 0$ then // action group is not already
           enabled
8          if $M(j) \geq a^-_{ij}\ \forall p_j \in {}^{\bullet}\mathbf{P}_{t_i}$ then // Enough preset
             markings, $i$ is the index of $t$
9            Enable the action group, disable all its transitions: $w_g$
               $= 1$, for All $t'$ in $g$ do $w_{t'} = 0$;
10     else // transition does not belong to an action
         group
11       if $M(j) \geq a^-_{ij}\ \forall p_j \in {}^{\bullet}\mathbf{P}_{t_i}$ then Transition is enabled:
           $w_t = 1$ ;
12       else Transition is disabled: $w_t = 0$ ;
13       if $w_t = 1$ And $x_t = 0$ then $\tilde{\tau}_t = \text{time} + \tau_t$, $x_t = 1$ ;
14 Sub-Algorithm2: Choose actions:
15   if any of the action groups is enabled then
16     get the description of the new RL state, $S'$, based on the RL
         environment
17     if $S' \notin \mathcal{S}$ then // $S'$ is new
18       $\mathcal{S} \leftarrow \mathcal{S} \cup \{S'\}$, $\mathcal{D}(S') = \{\}$ // append $S'$
19       for All $g$ in G do
20         if $w_g = 1$ And $c_g = 0$ then $\mathcal{D}(S') \leftarrow \mathcal{D}(S') \cup \{g\}$, $c_g = 1$ ;
21       Create actions, $\mathcal{A}(S')$, as combinations of transitions from
           each action group (Section 3.2)
22       $Q(S', a) \in_R \mathbb{R}, \forall a \in \mathcal{A}(S')$ // Initialize actions'
           Q-Values randomly
23     if $S'$ is not the first RL state in the episode then
24       $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$ // $S$
           is the previous state
25     $S \leftarrow S'$, reward: $R = 0$, choose $A$ from state $S$ using
         $\varepsilon$-greedy policy (Eq. (7)).
26     for All $t$ in $A$ do // The combination of
         transitions that represent the action
27       $w_t = 1$, $\tilde{\tau}_t = \text{time} + \tau_t$ // Enable and update the
           firing time
28 Sub-Algorithm3: Firing Conditions:
29   for All $t$ in T do
30     if $t \in g \backslash g \in$ G then // transition belong to any
         action group
31       if $w_t = 1$ And $\tilde{\tau}_t = \text{time}$ And no place is inhibiting transition
           then
32         $u_t = 1$, $w_t = 0$, $w_g = 0$, $c_g = 0$ // $g$ is the action
             group containing $t$
33       else
34         $u_t = 0$
35     else // The transition does not belong to an
         action group
36       if $w_t = 1$ And $\tilde{\tau}_i = \text{time}$ And no place is inhibiting transition
           then $u_t = 1$, $w_t = 0$, $x_t = 0$;
37       else $u_t = 0$;
38   Consider the effect of the fired probability transitions (take
       Eq.3 from [33])
39   Choose randomly one of each group of conflicting transitions
       to stay fired and unfire all others
40   Update the PN state: $\mathbf{M} \leftarrow \mathbf{M} + \mathbf{A}^T\mathbf{u}$, and get reward from the
       environment, $r$
41   Get rewards from the environment based on the updated
       state: $R \leftarrow R + r$
42 Main Algorithm:
43   for All Episodes do
44     time = 0
45     While time $\leq ET$
46       Sub-Algorithm1: Enabling Conditions
47       Sub-Algorithm2: Choose actions
48       if none of the transitions is enabled then Break the time
           loop;
49       else time $\leftarrow$ minimum firing time, $\tilde{\tau}$, of all enabled
           transitions ;
50       Sub-Algorithm3: Firing Conditions
51     Update the Q-Value of the final state-action pair:
         $Q(S, A) = Q(S, A) + \alpha[R - Q(S, A)]$
52     Update $\alpha$ and $\varepsilon$ based on Eq. (12), in Section 3.3
```

$$\alpha = \begin{cases} 1/n_u & \text{if } n_u < n_\alpha \\ \sigma_\alpha(n_e) & \text{if } n_u > n_\alpha \end{cases} \tag{11}$$

where n_u is the number of the Q-value updates, n_α is an input parameter that controls when to shift between methods, and $\sigma_\alpha(n_e) : \mathbb{N} \to \mathbb{R}^+$ is an exponential function controlling the decay of α based on the episode number n_e. The exponential function $\sigma_\alpha(n_e)$ is defined as follows:

$$\sigma_\alpha(n_e) = a + b\exp(-c \cdot n_e), \text{ with:} \tag{12}$$

$$\begin{aligned} a &= \alpha_{\min} \\ b &= \alpha_{\max} - \alpha_{\min} \\ c &= \ln[(\alpha'_{\min} - a)/b]/n_{ef} \\ \alpha'_{\min} &= \upsilon(\alpha_{\min} - \alpha_{\max}) + \alpha_{\max} \end{aligned}$$

where $\alpha_{\max}$ and $\alpha_{\min}$ are the values of α at the beginning and end of the decay respectively, and n_{ef} is the episode number at which the decay ends. Note that the output of this equation is maximum, i.e., $\alpha_{\max}$, at the beginning of the problem when the argument $n_e = 1$, whereas it reaches the minimum ($\alpha_{\min}$) when the argument equals n_{ef}. Because the decay is exponential, the outcome reaches its absolute minimum only at infinity. Indeed, the practical minimum, which is the value reached at n_{ef}, is controlled by υ which indicates how close the practical and the actual minimum are. This way of defining the exponential function wipe-out the need of guessing the value of decay because it takes the inputs directly as the required values at the beginning and end of the decay process.

Note also that the first part of Eq. (11) makes α to be the reciprocal of the number of times an action is chosen at the state. This will result in a Q-value equal to the average of all the previous expected returns calculated starting from that action, which neglects the effect of initial values chosen for the Q-values and consequently lets each Q-value shift quickly towards its converged solution since α will start by 1. The use of the reciprocal function should stop after a certain number of updates (namely, n_u in Eq. (11)) to prevent α from reaching very low values, which makes the effect of new expected returns negligible.

Finally observe that to balance between exploration and exploitation, Eq. (12) can also be used to define a function $\sigma_\varepsilon(n_e)$ for controlling the exploration rate ε; such that it takes different input parameters than

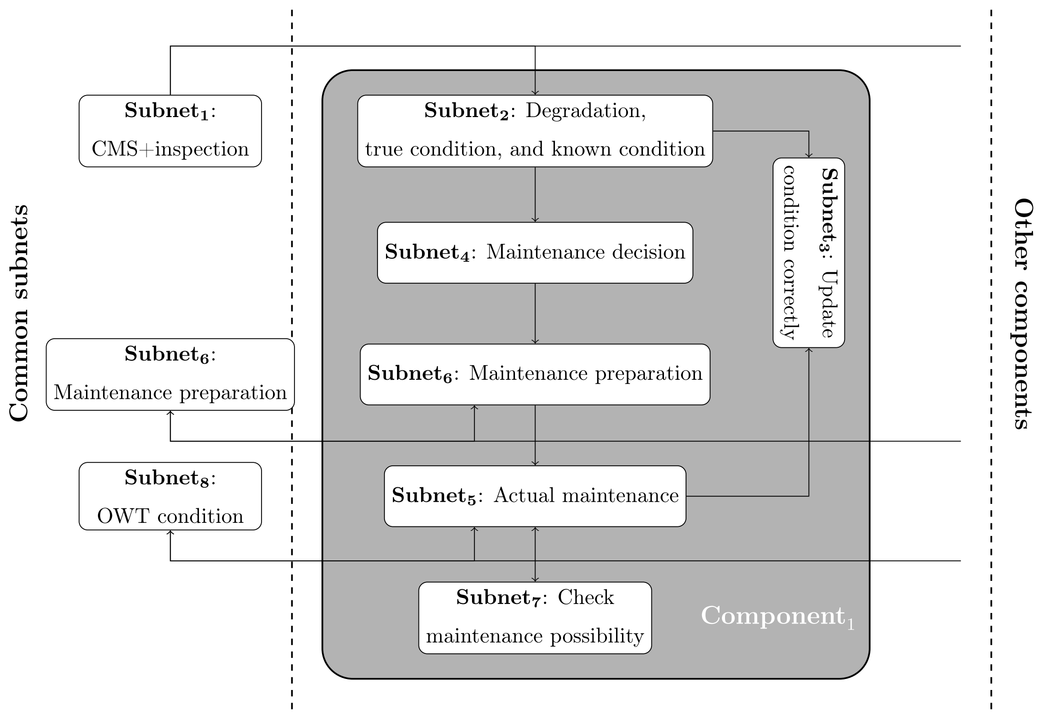

Common subnets
Subnet$_1$: CMS+inspection
Subnet$_6$: Maintenance preparation
Subnet$_8$: OWT condition
Subnet$_2$: Degradation, true condition, and known condition
Subnet$_4$: Maintenance decision
Subnet$_6$: Maintenance preparation
Subnet$_5$: Actual maintenance
Subnet$_7$: Check maintenance possibility
Subnet$_3$: Update condition correctly
Component$_1$
Other components

Fig. 5. Schematic description of the assemble using subnets for the *i*PN created to model the O&M of an OWT. The grey area is indicative of subnets which apply for all the OWT components, although the figure only illustrates the case of Component$_1$, for simplicity.

$\sigma_\alpha(n_e)$. The value of ε is also related to n_e, because the exploration needs to stop or to reach the minimum desired value before the end of the learning process, which is related to the total number of episodes, and this equation can give the full control by specifying the right inputs.

4. Case study: intelligent O&M of an OWT

The *i*PN method proposed in this paper is used to create and optimize an O&M model of an OWT. In this study, the OWT is modelled using subsystems which are composed of a number of components. The deterioration process of each component is divided into a number of states defined by specific thresholds. In this case study, these states differ from component to component, with a minimum of 2 and a maximum of 4 states. For example, if the *generation drivetrain* system is considered, the component starts in the normal state and then changes to a degraded state when excessive vibration time is detected. This vibration provides a pre-warning time until some heat is detected, which changes the state to critical condition. This condition lasts some time until the component reaches the failure condition. A condition monitoring system (CMS) is assumed to be installed in the OWT to provide updated information about each component's condition. In this work, the CMS is also considered as a component subjected to possible error and failure, whose condition is assumed to be detected only by performing an inspection or until it fails. If the CMS is in the degraded or critical state, the condition revealed is subjected to error, whereas when it fails, the component's condition can no longer be monitored, and the known condition of the component stays as the last known condition before the CMS failure. A list of all the subsystems with their corresponding components and conditions properties is provided in Table 3, given in Appendix.

4.1. iPN model for the OWT

This section presents the *i*PN model created to optimize the O&M of the OWT explained in the previous section. The model considers all the OWT components (20, in total), the CMS condition, the lifespan of the OWT, the electricity prices and running costs, the site accessibility, along with the weather conditions. The *i*PN model is schematically described by parts or *subnets* in Fig. 5, whereas the definition of each subnet is given in Figs. 6 to 10. The system has a tree structure with different subnets to perform different activities. Note that the shaded parts of the model represent the subnets of one specific component, *component*$_1$, namely the blade, and these subnets are repeated for each of the remaining components, although they are not represented in the paper for clarity and lack of space. Briefly, the structure of the system is composed of 20 components that are working and ageing concomitantly. It is assumed that the system can only be in two states, which are the working and failed, and that any failure of any component implies the failure of the whole system. The condition of each component is revealed by the CMS and periodic inspections, and the maintenance decisions are taken based on the condition of the whole system. Only CBM is considered, and the maintenance policy is fully controlled by the RL agent. The decision to repair a component is followed by logistic preparations, which is a component-specific activity, and then a travel to the sight after finding a good weather window. After that, maintenance is done for the selected components if the conditions allow that. Note that the sequence of the activities can be better understood from the connections of the nodes of each subnet and this is explained in the sections below. A description of the nodes of the *i*PN is provided in Table 4. Finally observe that the nodes in the shaded areas are given names with subscripts (i, u, m, and c) to make finding their subnets easier, and also for differentiating them from the common nodes, which are left with numerical subscripts.

In the model, the normal, degraded, critical, and failed states are represented in subnet$_2$ (Fig. 6) by places p_{i1}, p_{i2}, p_{i3}, and p_{i4}, respectively. Transitions t_{i1}, t_{i2}, and t_{i3} represent the degradation steps to evolve between the aforementioned states. The time for each transition is assumed to follow a Weibull distribution whose parameters are based on the values used by Le and Andrews [31]. The condition of CMS is represented by the places p_1 to p_4 from Subnet$_1$ (Fig. 6, upper panel), which reveals the known condition by the places p_{i5} to p_{i8} from Subnet$_2$. Places p_{30} to p_{33} are used to represent the known condition of the CMS, since it can be different from the actual one (represented

Table 1
Description of the different repair types with their time and monetary requirements. The wait weather time is modelled as a Weibull distribution characterized by shape parameter β, and scale parameter η.

Type	Description and requirements	Logistic time (h)	Travel time (h)	Wait weather time (weeks)	Repair time (h)	Typical rate (£)
1	Heavy component, require external crane and Crane ships	500	3	$\beta = 3.1$, $\eta = 6$	70	210,000
2	Heavy component, require internal/external crane (>800–1000 kg) and Jack-up barges	160	3	$\beta = 3.5$, $\eta = 3$	50	120,000
3	Small parts, require internal crane (<800–1000 kg) and Jack-up vessels	48	3	$\beta = 3.4$, $\eta = 2$	10	45,000
4	Small parts, inside nacelle, require Crew Transfer Vessels	24	3	$\beta = 3.3$, $\eta = 1$	3	15,000
5	Small parts, outside nacelle, require Crew Transfer Vessels	8	3	$\beta = 3.2$, $\eta = 6$	3	21,000

by places p_1 to p_4) because these conditions cannot be detected until performing an inspection or until it fails.

The marking of places p_{i12} or p_{i9} describes if the state detection is erroneous or not, respectively. If the CMS is in normal state, t_{i7} will mark p_{i9}, whereas if the CMS is in degraded or critical state, then the firing of t_{i8} or t_{i9} has a probability of marking p_{i12}. The probabilities for t_{i8} to mark p_{i12} and p_{i9} are 0.2 and 0.8, respectively, whilst the probabilities for t_{i9} to mark p_{i12} and p_{i9} are set to 0.7 and 0.3, respectively. The update of the component's condition in case of error is done through the probability transition t_{i11}, and in the absence of error, through the Subnet$_3$ shown in Fig. 7 after marking place p_{u1}.

Additionally, the model assumes that the inspection of the OWT is performed regularly, every 6 to 12 months [31], and also that it gives an update of the known conditions of all components with no error. This is modelled by firing the timed transition t_4, which marks p_{u1} and p_{34} to update the condition of the components and CMS, respectively. When the expected inspection occurs, a decision has to be taken regarding the repair of the CMS, provided that it is not in a normal state. This decision is trained through the RL action group g_1 by marking place p_{28} through one of the transitions from t_{33} to t_{35}.

Every time the known condition of a component changes due to degradation, monitoring, or inspection, a decision has to be taken regarding which combination of deteriorated components has to be repaired according to the new state of the environment. This is represented by marking place p_5, which activates the *i*PN Subnet$_4$, shown in Fig. 8. The decision-making process and the details of the RL input parameters for training decisions are described in the following section.

4.2. Reinforcement learning for repair decisions on the OWT's components

The proposed model in this work assumes that the system's condition changes when the known condition of any component changes, thus a maintenance decision is required. This decision-making process is modelled through firing transition t_5 in Fig. 8, which marks p_{m1} to check if a repair is needed. If the known condition of the component is normal, or if the component was chosen to be repaired previously, p_{i5} or p_{m5} will fire t_{m3} or t_{m1}, respectively, to unmark p_{m1}. If these two conditions do not happen, then t_{m2} will produce a token to p_{m3} to cause the adoption of a decision through the action group g_2 about whether a repair was required. Action group g_2 for all deteriorated components are enabled simultaneously to make the corresponding decision for these components at the same time, which is considered as one RL action, as was defined in Section 3.2.

Remark 2. As indicated in Section 2.3, the decisions are taken based on the state of the defined RL environment. In this case study, the environment is defined by: the known condition (p_{i5} to p_{i8}), the repair decision place (p_{m5}), and the repair start place (p_{m8}) for each of the components; the known conditions of the CMS (p_{30} to p_{33}); and the number of failures of the system (p_{36}). The true conditions of the components cannot be assumed to be part of the environment because they are not available to the user, and considering them excludes the effect of CMS and inspection. Based on the specifications of the RL environment, a state is automatically created (if it was not already created) with its available actions each time a decision is required.

Once the decision to repair a component is made, further repairs of the same component are not allowed until the current repair action has been finished or cancelled. This is modelled by marking p_{m5}, which inhibits t_{m2}. Then, one of the transitions t_{m9} to t_{m10} is fired according to the component's known condition to start the repair preparation process according to the required type of repair. The *i*PN Subnet$_6$ shown in Fig. 9 describes the repair preparation process, whose rationale is explained in the following section.

4.3. Description and modelling of repair activities

In this case study, the repair actions are classified according to the size of the component and the equipment required for the repair. Table 1 shows the types of maintenance adopted here based on the maintenance categories used in the industry [66]. The table also shows the requirements, amount of time, and estimated costs of each maintenance type. Note that maintenance types no. 4 and 5 are devoted to repairing small parts outside and inside the nacelle, respectively. They both require small and fast vessels named *crew transfer vessels* to transfer technicians, other personnel, and equipment spares to the OWT's site for minor repairs or inspections. Maintenance types no. 2 and 3 require jack-up rigs that allow transporting the unit to the desired location, and movable legs that allow raising the hull over the sea surface. Maintenance type no. 1 requires ships equipped with pedestal-mounted or sheer-leg cranes specialized in lifting heavy loads to the nacelle. Maintenance type no. 3 can use self-propelled rigs named jack-up vessels, whereas maintenance type no. 2 requires non-self-propelled vessels, named jack-up barges, with support ships that tow them to the working position. The aforementioned types of maintenance require internal (<800 kg) and external (>800 kg) cranes respectively. The costs reflected in Table 1 comprise those related to the type of repair, which is considered before doing the actual maintenance. The type of repair, the required maintenance action, and the cost for such action for each of the components at their different degradation states are listed in Table 3, given in Appendix.

In this work, opportunistic maintenance can be an option, thus a repair action is triggered when there is more than one component demanding maintenance. In such a case, the type of repair adopted from Table 1 is the one that corresponds to the more demanding component, such that the requirements of this component can cover the needs for repairing the remaining less-restrictive components. However, the OWT's components have different degradation rates, thus waiting until several components need repair can imply the failure of some of them or a decrease in the OWT's productivity. Here lies the importance of RL to decide on the combination of components to be repaired based on the state of the environment, so that optimal performance is achieved.

For the modelling of the repair, the process has been divided into 4 steps, namely: *planning and logistics preparation, waiting for good weather, travelling and accessing the site*, and *actual repair*. In this work, the time required for each of the repair processes, which is dependent on the type of repair, is shown in Table 1. Also, the common processes between components, which are the ones before the actual repair, are denoted by the term *repair preparation*.

Fig. 9 provides the definition of the Subnet$_6$ from the *i*PN used to model the repair preparation processes. Observe that the model starts by firing one of the transitions t_{m9} to t_{m11} according to the condition

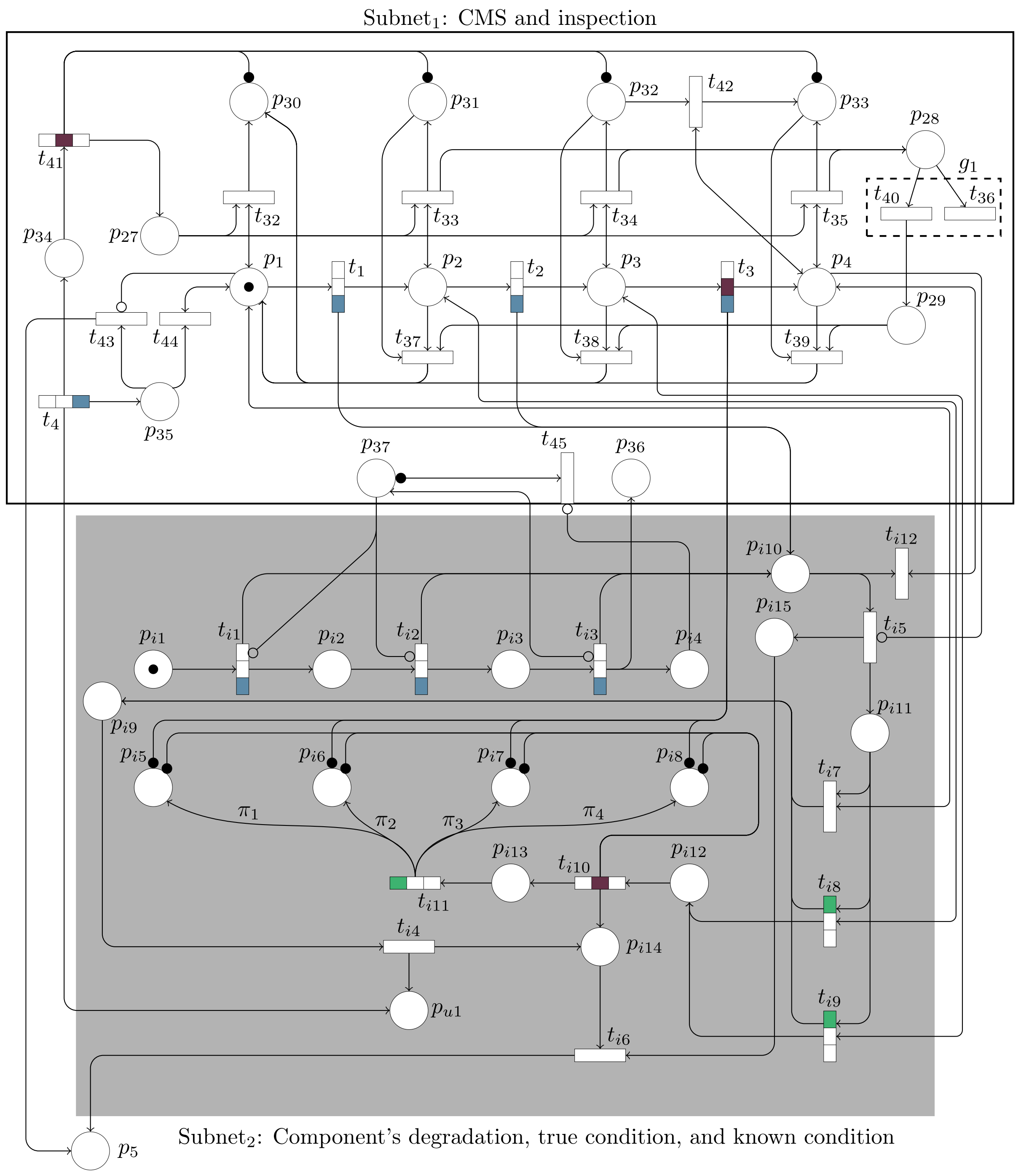

Subnet$_1$: CMS and inspection
Subnet$_2$: Component's degradation, true condition, and known condition
p_{30} p_{31} p_{32} t_{42} p_{33} p_{28} g_1 t_{40} t_{36}
t_{41} p_{34} p_{27} t_{32} t_{33} t_{34} t_{35}
p_1 t_1 p_2 t_2 p_3 t_3 p_4 p_{29}
t_{43} t_{44} t_{37} t_{38} t_{39} t_4 p_{35}
p_{37} t_{45} p_{36}
p_{i10} t_{i12} p_{i15} t_{i5}
p_{i1} t_{i1} p_{i2} t_{i2} p_{i3} t_{i3} p_{i4}
p_{i9} p_{i11}
p_{i5} p_{i6} p_{i7} p_{i8} t_{i7}
π_1 π_2 π_3 π_4
p_{i13} t_{i10} p_{i12} t_{i11} t_{i8}
t_{i4} p_{i14} p_{u1} t_{i9} t_{i6} p_5

Fig. 6. Illustration of the *i*PN subnets for modelling operation, inspection, and condition monitoring actions.

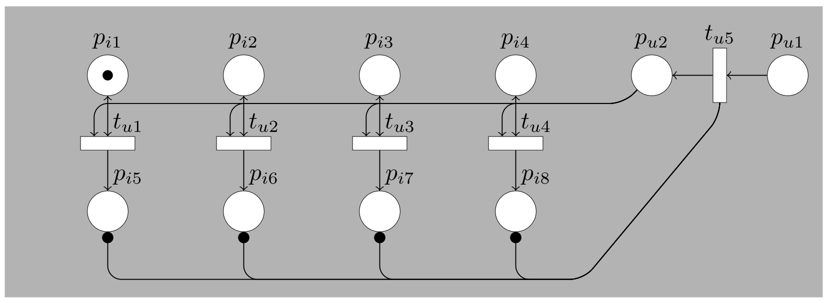

p_{i1} p_{i2} p_{i3} p_{i4} p_{u2} t_{u5} p_{u1}
t_{u1} t_{u2} t_{u3} t_{u4}
p_{i5} p_{i6} p_{i7} p_{i8}

Fig. 7. *i*PN (Subnet$_3$) used for component's condition updating.

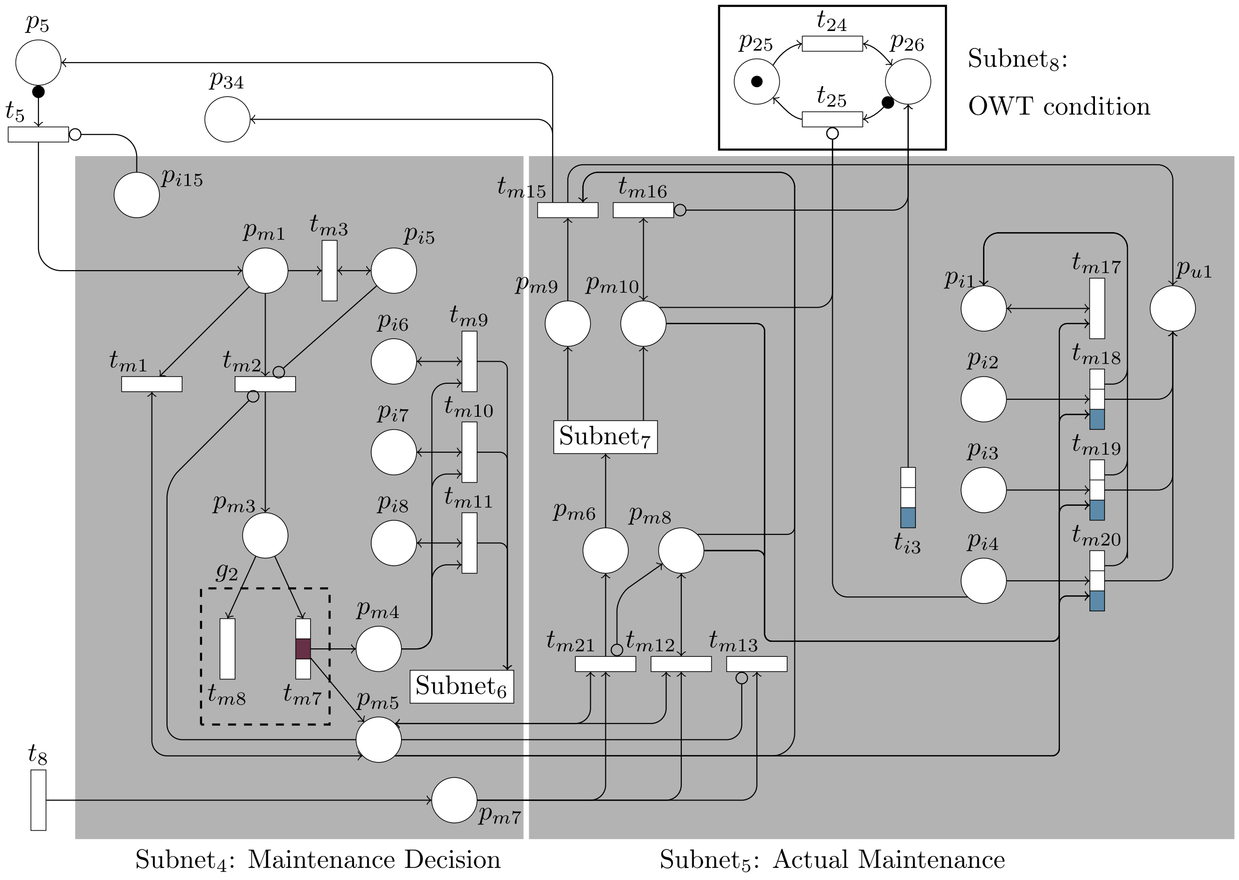


Fig. 8. Illustration of the *i*PN subnets for modelling maintenance decisions and actions.

of each component, indicated by a token in p_{c1}, p_{c2}, or p_{c3}. Next, the logistic preparation starts according to the type of the required maintenance (indicated by marking any of the places p_9 to p_{13}), and then the model indicates that the logistic preparation is undergoing (p_6), updates the counters of the number of components that require each type of repair (p_{19} to p_{23}), and triggers the maintenance preparation (p_{24}). Then, according to the required type(s) of repair (p_{19} to p_{23}), part (or all) of the t_9 to t_{13} transitions are enabled. These transitions represent the time for logistic preparation. Note that if more than one component needs logistic preparation with the same type of maintenance, that preparation happens only once, which is modelled by resetting the logistic place after the logistic transition is fired. After the logistic preparations for all the types of maintenance have finished, transition t_6 is fired to trigger the *waiting for good weather* state (represented by marking place p_7).

In this model, the more demanding types of maintenance that can fulfil the needs of the less demanding maintenance types, are selected. Thus, the *waiting for good weather* state will be based on that selected type, so that the subnet p_{14} to p_{18} and t_{14} to t_{18} will guarantee that only one of the transitions will be fired based on the most demanding type of repair. After that, p_8 will be marked producing the firing of t_8 after the *travel to the site* time has passed. Firing t_8 will mark p_{m7} in Fig. 8, which indicates that the OWT site is reached and actual repair is undergoing.

In some cases, the condition of a component that was chosen to be repaired can change while preparing for the repair. In this case, the repair preparation for that component should be stopped. Place p_{m12} is marked every time a condition of a component is changed. Accordingly, t_{m25} is fired if the repair preparation is undergoing (p_{24}), it was decided to repair this component (p_{m5}), and the component is not already under repair (p_{m8}). This will mark p_{m11}, which will fire one of the transitions t_{m22} to t_{m24} according to the known repair type when this was decided. That will update the number of components that require each type of repair (places p_{19} to p_{23}), and cancel the decision on repairing that component (unmark p_{m5}). When any of the counters p_{19} to p_{23} is emptied, the corresponding transition t_{27} to t_{31} is fired to cancel the logistics of that type of repair, provided that it is undergoing, or the corresponding transition t_{19} to t_{23} is fired to cancel the *wait for good weather* of that type of repair. When all the counters are emptied and preparation for a repair is undergoing, it means that the preparation actions of all components are cancelled, so transition t_{26} is fired to cancel that repair preparation.

After the site is reached (p_{m7}), three courses of action exist:

1. the component was decided to be repaired (p_{m5}), but the component is already under repair via another maintenance travel launched before. Then, t_{m12} will unmark p_{m7};
2. the component was not decided to be repaired (unmarked p_{m5}), so t_{m13} will also unmark p_{m7};
3. the component was decided to be repaired (marked p_{m5}) although it is not under repair yet (unmarked p_{m8}), thus t_{m21} will fire to continue the process by marking p_{m8} and p_{m6}.

In some cases, the actual repair of the component cannot take place. This might happen if the preparation for the repair is done based on erroneous conditions because of a CMS error, or if the component degraded just while travelling to the site. This situation is addressed here by using the *i*PN subnet shown in Fig. 10, which is used to check if the repair can continue by comparing the known condition when the preparation of the repair started and the true condition after reaching the site. This *i*PN subnet is activated after p_{m6} is marked and it shows twelve scenarios represented by transitions t_{c1} to t_{c12}. The cases indicating that the repair can continue are connected to transition p_{m10}, whilst the rest are linked to p_{m9}. The connections of transitions with places p_{m10} and p_{m9} differ from component to component based on the

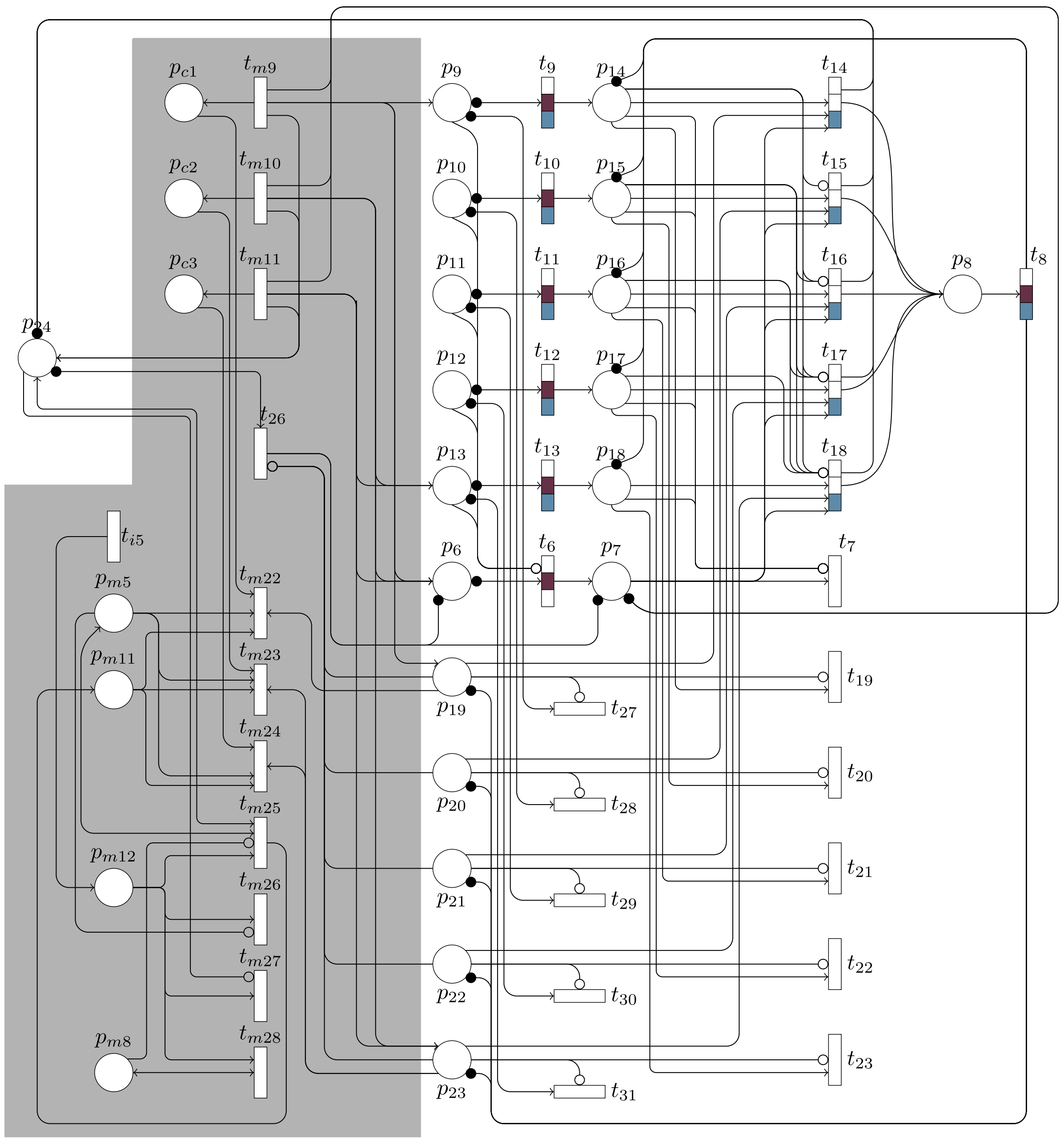


Fig. 9. *i*PN (Subnet$_6$) for repair preparation.

number of conditions, the type of maintenance, and the action required at each condition. The particular case corresponding to each component is studied separately and their corresponding scenarios are shown in Table 2. It is also assumed that, if the type of maintenance actions required is the same regardless if full or partial replacement is needed, then the maintenance team will prepare for full replacement since it does not require any additional cost.

After finding that the repair cannot continue (p_{m9}), t_{m15} is fired to update the known condition of the component (p_{u1}), update the known condition of the CMS and decide on repairing it (p_{34}), and send a new maintenance team to repair the deteriorated components (p_5). On the other hand, if the repair can continue (p_{m10}), t_{m16} is fired to indicate the shutdown of the OWT to perform maintenance by marking p_{26}, and one of the transitions t_{m17} to t_{m20} will be enabled according to the true condition of the component to model its actual repair.

Note that t_{m17} is an instant transition just to pass the token if the true condition is normal. This happens if the CMS wrongly detected a deteriorated state of the component. After the actual repair ends, it is assumed that the component returns to its pristine-normal condition. Accordingly, p_{m10} will be unmarked, which will allow firing t_{25} to get the OWT back to its working condition (p_{25}). Also, it will unmark places p_{m5} and p_{m7} to indicate that the component is no longer under repair, and will mark p_{u1} to update the known condition to the normal condition.

4.4. Reinforcement learning inputs

As explained in Section 3.3, $\sigma(n_e)$ is proposed to control the decay of the learning rate α and the exploration rate ϵ, by creating $\sigma_\alpha(n_e)$ and $\sigma_\epsilon(n_e)$ functions respectively. Here, the end of decay n_{ef} for both functions is set to $0.95n_t$, where n_t represents the total number of episodes, set to $n_t = 100'000$. Besides, ϵ is proposed to start by $(\sigma_{\max})_\epsilon = 1$ (fully exploratory), and to end by $(\sigma_{\min})_\epsilon = 0.001$ to keep a very low probability of exploring the environment, which does not affect the

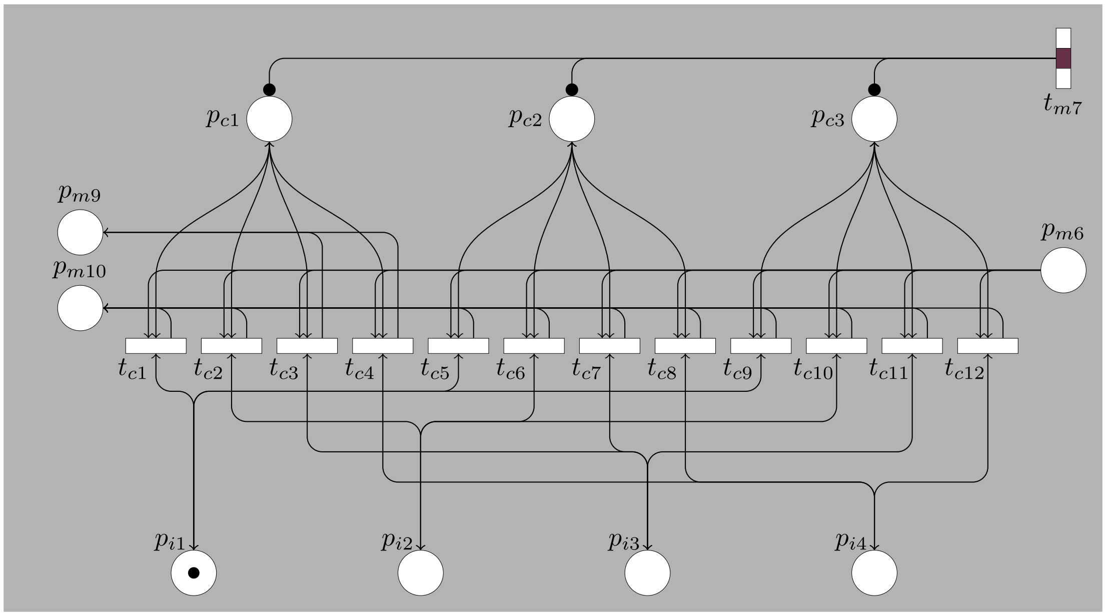


Fig. 10. *i*PN (Subnet$_7$) used to check if the maintenance can continue based on known condition of any component.

Table 2
Evaluation of the possibility to continue the repair of the OWT components based on the possible scenarios, which arise as combinations of the true condition after reaching the site and the known condition when preparing for maintenance.

Transition	True condition	Known condition	Component number (refer to Table 3)																			
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
t_{c1}	Normal	Degraded	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
t_{c2}	Degraded	Degraded	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
t_{c3}	Critical	Degraded	✗	✗	✗	✗	✓	–	–	✓	–	–	–	–	✗	✗	✓	✗	✗	✗	✓	✓
t_{c4}	Failed	Degraded	✗	✗	✗	✗	✗	✓	✓	✗	✓	–	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗
t_{c5}	Normal	Critical	✓	✓	✓	✓	✓	–	–	✓	–	–	–	–	✓	✓	✓	✓	✓	✓	✓	✓
t_{c6}	Degraded	Critical	✓	✓	✓	✓	✓	–	–	✓	–	–	–	–	✓	✓	✓	✓	✓	✓	✓	✓
t_{c7}	Critical	Critical	✓	✓	✓	✓	✓	–	–	✓	–	–	–	–	✓	✓	✓	✓	✓	✓	✓	✓
t_{c8}	Failed	Critical	✓	✓	✓	✗	✗	–	–	✗	–	–	–	–	✓	✗	✓	✓	✓	✓	✗	✗
t_{c9}	Normal	Failed	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
t_{c10}	Degraded	Failed	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
t_{c11}	Critical	Failed	✓	✓	✓	✓	✓	–	–	✓	–	–	–	–	✓	✓	✓	✓	✓	✓	✓	✓
t_{c12}	Failed	Failed	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

calculations of the Q-values. This is done along with the control of α, which starts with $(\sigma_{\max})_\alpha = 0.1$ after performing $n_u = 10$ updates and reaches $(\sigma_{\min})_\alpha = 0.001$ to reduce the effect of fluctuations and keep the effect of old updates on averaging the Q-values since the policy becomes more trusty by the end of the solution. The value of υ is specified as 0.99 for both α and ε.

Under this configuration, the resulting RL algorithm aims at harvesting the maximum possible wind energy while minimizing the O&M costs. The rewards, expressed in monetary terms (by assuming electricity rates ER), consider the average annual energy production in kWh for an OWT, which is calculated based on its capacity P expressed in MW units, and by taking into account the capacity factor CF, as follows:

$$\begin{aligned} E[\text{kWh}] &= CF \cdot P\ [\text{MW}] \cdot 1\ [\text{year}] \\ &= CF \cdot P\ [\text{kW}] \cdot 1\text{e}3 \cdot 8760\ [\text{h}] \\ &= 8.76\text{e}6 \cdot CF \cdot P\ [\text{kWh}] \end{aligned} \tag{13}$$

In the last equation, E is the average annual production. Here, the lifespan of the OWT is assumed to be 25 years. To account for the downtime of the OWT, the model is made to produce energy only when it is in the working state (p_{25} marked). Accordingly, the average revenues during a period p is calculated by multiplying the ER by the energy produced in that period, as follows:

$$R = \begin{cases} E \cdot ER \cdot p & \text{if } M(p_{25}) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where period p is expressed in years. Today's new OWT capacities, P, ranges from 3 to 8 [MW] [67] with a capacity factor, CF, ranging between 40% to 50% [68]. In this study, it is assumed that the OWT capacity is 3 [MW] with a capacity factor equal to 45%, whereas the electricity rate is set to $ER = 0.2$ [£/kWh] (taken from average rates in the United Kingdom provided by the Global Petrol Prices Website [69]). Notwithstanding, revenues are not considered directly for calculating the rewards because the Q-learning is a one-step bootstrapping method, which works better if the change between consecutive states is recognizable and significant [43]. Since the OWT generate revenues most of the time, considering them as rewards will cause small differences between states. Hence, the losses of revenues are considered to be negative rewards, and they are calculated by subtracting the actual revenues from the maximum possible revenues of the OWT.

On the other hand, O&M costs are also represented by negative rewards. These costs include the actual maintenance costs provided in Table 3 and linked to transitions t_{m18} to t_{m20} of each component, and t_{37} to t_{39} for the CMS. The O&M costs also include the preparation for maintenance costs provided in Table 1, which are linked to transitions t_{14} to t_{18}, along with the inspection costs, which are estimated in £40,000 per inspection and linked to transition t_4.[2] Note also that all the steps before the actual repair takes place are included in the preparation costs. These costs can be shared by more than one component that requires the same type of maintenance. Finally, the downtime of the OWT can be shortened if the maintenance of more than one component is done at the same time. Thus, the goal of our *i*PN algorithm is to find the best condition and time to undertake the maintenance actions for each of the components, thus the system downtime and the costs are minimized while increasing the revenues.

5. Results

This section shows the results of the simulations using the proposed *i*PN algorithm to optimize the O&M schedule of the considered OWT case. Here, the learning process is divided into intervals of 500 episodes each to calculate the average value of some variables of interest, along with the uncertainty bounds. Fig. 11 shows the variation of the mean and the uncertainty bounds of different operation variables, as a function of the episode number. Observe that the uncertainty bounds at the beginning of the learning process are high because at that stage the policy is fully exploratory. Also, note that the policy at the end of the learning process is deterministic and based on full exploitation, thus the uncertainty bands are much smaller. Irrespectively, the remaining, but small, uncertainty that can be observed towards the end of the process appears because of the variability of the problem, e.g. stochastic degradation rates, that cannot be eliminated.

Fig. 11a shows the total reward per episode. Observe that, since rewards have been expressed in terms of costs and losses, thus the total reward is always a negative quantity. The figure shows that the average total reward at the end of the learning process is almost equal to the upper uncertainty bound, which is the maximum that it can reach, and also constitutes an indication that the RL was successful in finding the optimal policy, i.e., the one that increases the rewards.

The increase in the total rewards is also reflected on different O&M variables like the productivity losses (Fig. 11b), revenues (Fig. 11c), costs (Fig. 11d), turbine availability (Fig. 11e), and the number of failures (Fig. 11f). Note that the RL agent within our proposed *i*PN algorithm has been able to increase the revenues and the availability to reach their maximum possible values, and also to decrease the number of failures and losses to almost 0. However, observe that it was not able to completely eliminate the costs, due to the fixed costs of the periodic inspection. Another reason for the last relies on the necessity to perform maintenance actions to avoid downtime and failure.

Besides, the results obtained from the *i*PN algorithm reflect that the costs are very sensitive to the uncertainty, probably because of the stochastic degradation rates of the components, which result in different patterns of components' conditions for each episode, thus requiring different maintenance actions and costs. Moreover, the results indicate that stochastic deterioration can also have an effect on downtime because of the different times required for maintenance preparation and actual action. The time to prepare for maintenance can contribute to downtime if the component is in a failed state. However, this is avoided once the failure of the OWT is avoided. Yet, the downtime caused by the actual repair cannot be avoided, but its effect on the availability is minimal since it requires a very short period of time.

Finally, the results in Fig. 11 also show that the effect of the costs is minimal on the total rewards because the decrease in the losses is almost equal to the increase in the total rewards, and the uncertainty in the costs barely adds uncertainty to the total rewards.

[2] Roughly, the O&M costs of offshore wind turbines are around €0.02 per KWh [70], and the electricity rates around €0.2 per KWh, hence the costs to revenues ratio is around 10%. A simulation was done while following periodic maintenance, and it was found that to preserve that ratio adopted by the industry, the O&M costs should be about 3 times the values provided by Le and Andrews [31]. Thus, these costs were adopted.

6. Discussion

This section provides a discussion about the results obtained above. Indeed, the first aspect to highlight taking the results at hand, is that increasing the revenues of an OWT, which is done by decreasing the negative rewards arising due to downtime losses, is much more important than decreasing the maintenance costs. Our simulations indicate that the RL agent wisely chose to undertake costly actions in the short term in pursuit of achieving greater profits in the long run. This is why it is always important in RL to define the rewards in terms of ultimate goals, and not based on what is expected to be beneficial for increasing the ultimate goal. That also remarks the need for expressing all types of rewards in the same unit, e.g., GBP in our case, otherwise, they might be unbalanced, which can drive the policy to avoid making maintenance in order to decrease the negative rewards of the costs.

To better discuss the role of the RL agent within the proposed *i*PN algorithm, an analysis has been carried out based on a simulation of 2000 episodes following the final policy achieved with $\varepsilon = 0$ (without any exploration). The results are illustrated in Fig. 12 by showing the 46 most frequent RL states and their corresponding actions for a selection of components, namely components 21, 1, 2, 18, 19, and 20, as indicated in Table 3. In this exercise, each component can be in one of 6 possible states, which are: normal, degraded, critical, failed, under preparation, or under repair, indicated in Fig. 12 by the legend.

The figure is divided into two parts, the upper one describes the RL states whereas the lower one describes the corresponding actions. An index is assigned to each of the states for easier interpretation, and these indexes are labelled horizontally above the upper part (1,6,11, . . .). For example, the conditions of components 21, 1, 2, 18, 19, and 20 at state number 11 are: normal, normal, degraded, normal, degraded, and normal, respectively, which is shown by the coloured column under number 11. This state has two components in a degraded state, which are components 2 and 19. Accordingly, four actions exist at this state that are the combinations of decisions that can be taken for each of the components. These actions are: repair–repair, repair–no repair, no repair–repair, and no repair–no repair decisions taken for components 2 and 19 respectively. The Q-values for each of the four actions are: $-8.8 \cdot 10^5$, $-6.8 \cdot 10^5$, $-11.6 \cdot 10^5$, and $-7.2 \cdot 10^5$, respectively, thus the optimal action, which is the one with the highest Q-value, corresponds to the second one, namely to repair the first and not to repair the second component. This action is illustrated in the lower part of the figure by the coloured column no. 11, which describes the action as a group of decisions, namely: no decision, no decision, repair, no decision, do not repair, and no decision for components 21, 1, 2, 18, 19, and 20 respectively. The "no decision" status is given for the components that do not require a decision about being repaired, i.e., they are not repaired.

The analysis and interpretation of the optimal policy figure (Fig. 12) also provide some hints on how the rewards have been optimized through our proposed *i*PN. The first observation is that the algorithm favours the repair of components 1, 2, and 18, but not the repair of components 19 and 20 in most of their deteriorated states. The main difference between these two groups of components is the deterioration rate, which is represented by the size parameter, η (refer to Table 1). This parameter is larger for components 19 and 20, which makes their probability of failure low before the end of the OWT life. Thus, the algorithm enables the use of these components with free maintenance during the lifespan unless they are about to fail. On the other hand, the preferred decision for components 1, 2, and 18 was to repair once the deterioration occurs, and this is because repairing at a degraded state is much cheaper than doing a major repair at a critical or failed state. Besides, the change of the condition to a degraded state takes much more time than the change to critical or failed. This means that not repairing in a degraded state for these components can only add a short period of service before the repair becomes unavoidable. In addition to that, if a component fails, it causes additional downtime, the

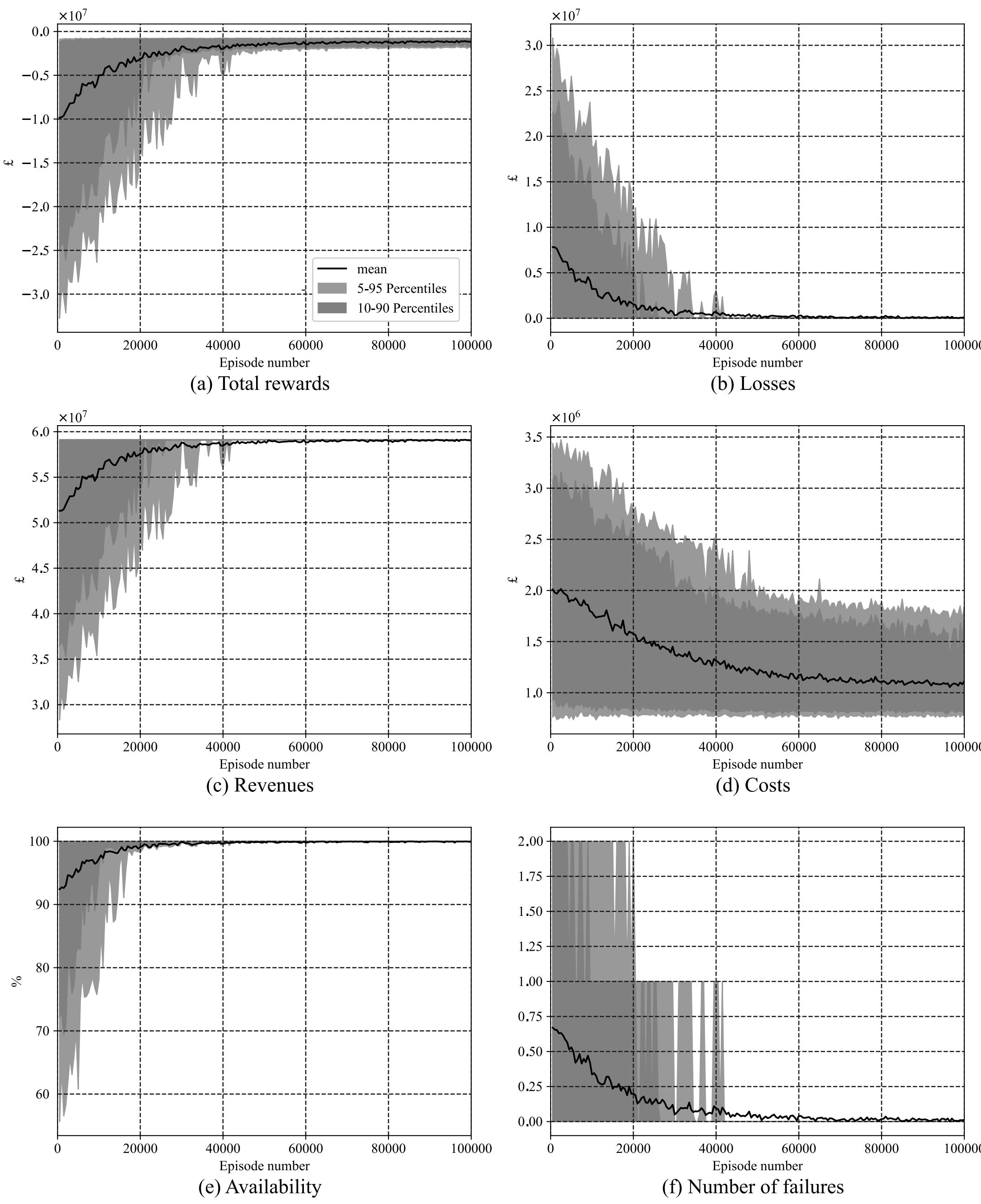


Fig. 11. Results of the simulation of the *i*PN algorithm for the OWT case study, with the indication of the learning for six O&M variables as a function of the number of episodes.

highest maintenance costs, and decreases the efficiency of the turbine. Indeed, failure causes the longest downtime because of the need to stop the OWT from producing energy for the time taken to detect the failure, prepare for maintenance, travel to the site, and make the actual maintenance. The downtime to repair at any other condition only includes the time of the actual maintenance, because the OWT can still work during the preparation steps. Precisely, our algorithm reveals that it is preferred to always maintain in good condition the components that can reach failure.

Observe as well that the case of the CMS is similar to that of components 19 and 20 because it has a slow deterioration rate. Thus, paying the costs of repairing the CMS at the end of the service life may not be cost-effective because the payback period of that action may be greater than the remaining useful life (RUL). In addition to that, it is not important to keep the components in perfect condition in the last years of the service life because they are going to be disposed of in just a few years. Therefore, knowing the accurate condition of the components via the CMS is not that important anymore, and inspections can be enough in that period. In fact, it can be noticed that the decision was to not repair the CMS for most of the states.

The RL agent was able also to do interesting work by understanding when to trust or not the information coming from the CMS. For example, states 37 and 39 show a failure of component 18, but the RL decision was not to repair this component in these states. At state 39, the condition of the CMS was critical, so it can be easily concluded that the revealed failure was an error. However, at state 37, the condition of the CMS is degraded, which means that it was more probable that the CMS revealed the correct condition, but the RL took the decision not to trust the CMS. This was because the final policy was to repair component 18 when any deterioration is detected, which makes it very rare for this component to reach a failed state. Accordingly, the algorithm learnt that the component was not failed and the CMS was mistaken. Not repairing at that condition means postponing the decision until the next inspection happens, which reveals the true state of the component.

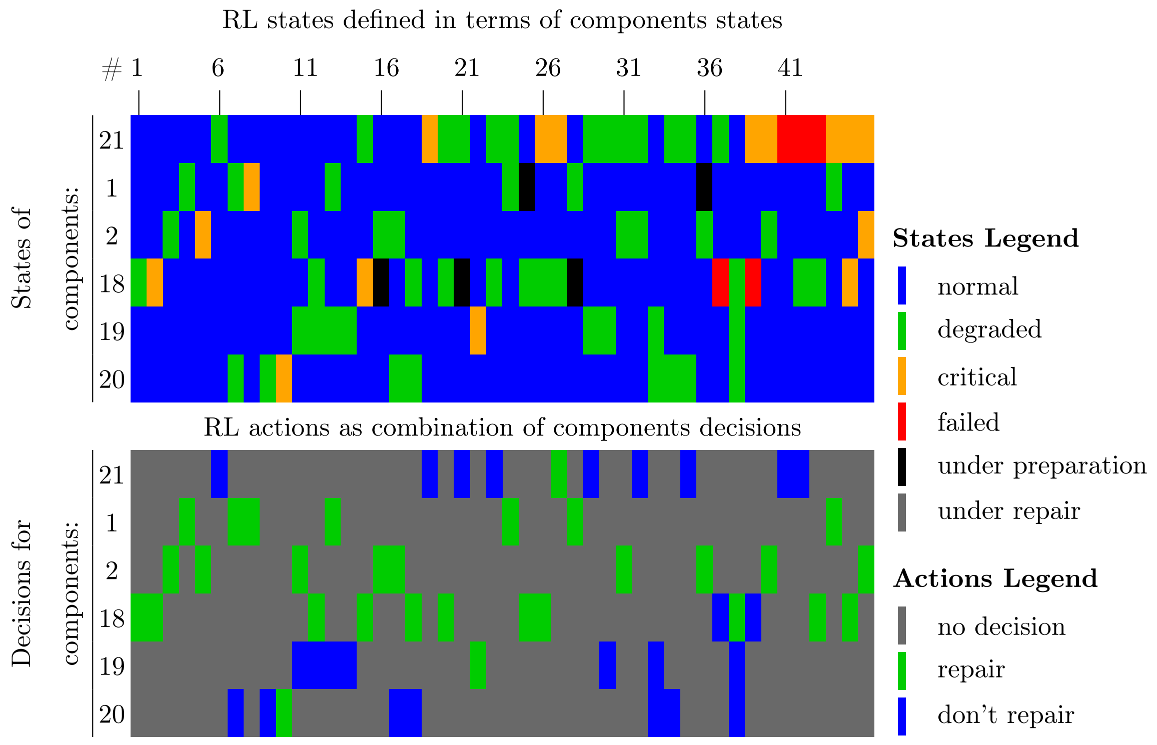


Fig. 12. The most repeated RL states when following the optimal policy and the corresponding decisions at each state.

RL was able to use the combinations of the component's states as an indicator of the RUL and to use it for making decisions.

As a curiosity, and to corroborate the results from the *i*PN algorithm, note that although the decision was not to repair components 19 and 20 for most of the states because they have low probabilities of failure, the decision at states 10 and 22 was to repair these components. This is because the condition of components 19 and 20 was critical at these states, which gives more chance to have failure. Thus, a repair was made to completely eliminate the probability of failure no matter what the maintenance costs are. Based on the policy results, and by comparing the values of the revenues and the costs (Fig. 11), it can be concluded that avoiding downtime and failure is more important than saving maintenance costs. In light of this information, it was decided to follow the strategy of repairing when any deterioration is detected for the schedules of the components that were not optimized by RL. This strategy with the RL policy formed the decision support system (DSS) for the whole OWT.

Finally, to evaluate the DSS, the *i*PN model was simulated based on its decisions, based on corrective maintenance, and based on periodic maintenance; then, the three simulations were compared. The corrective maintenance was made by enabling the repair decision of each component only when the known condition of the component failed. On the other hand, periodic maintenance was made by disabling the CMS and making minor repairs for each component periodically. Accordingly, the net profit per lifespan, calculated by subtracting the costs from the revenues, was £56.1 MM, £11.6 MM, and £29.3 MM for the DSS, the corrective one, and the periodic strategy, respectively. The DSS has the highest profit because it has resulted in very high availability, around 99.4%, and a very low probability of failure, around 2%. The costs spent for the DSS case are £2.48 MM, which represent only 4.42% of the profit, so using the RL to further optimize the maintenance strategy by considering all the components instead of the five in the RL decisions can result in a very slight improvement. This implies that the current strategy of the DSS can be considered as an approximate optimum strategy, which outperforms the conventional ones.

The current study uses a tabular reinforcement learning method, which is Q-learning. Tabular methods can reach exact solutions when the state space is discrete and not too big, but they are unfeasible for enormous or continuous state spaces. Thus, if any of the variables describing the RL environment is continuous (e.g. components' conditions based on continuous damage features) it is necessary to discretize these variables before applying the tabular methods. However, discretization can result in a huge state space, so for such cases, it is better to use *function approximation* RL methods, which rely on finding a general approximate value function using any of the supervised learning techniques rather than estimating the value for each particular state–action pair separately.

7. Conclusions

A novel general methodology has been provided for combining RL with HLPN to form an intelligent PN model. The method, which has been named *i*PN, allows upgrading any conventional PN model, regardless of the number of states or actions, by gathering groups of transitions describing the conflicting actions in sets called action groups. The proposed methodology has been formulated in a general manner, however, this paper gave special focus to O&M problems, by providing specific treatment of practical O&M aspects within the *i*PN formulation. The method was used to autonomously find an optimum policy for the O&M of an OWT while considering different influencing factors. The following are some concluding remarks:

- the *i*PN algorithm has been able to deal with a final policy which makes an OWT availability equal to 99.4% while minimizing the maintenance costs, thus increasing the profits to the maximum;
- the *i*PN algorithm was able to learn when to trust the condition monitoring system based on the overall condition of the system. Also, it was able to recognize the probability of failure of each component based on their pattern of conditions and take maintenance actions accordingly;
- the created model was able to experience all the possible state–action pairs and evaluate each of them separately, creating a successful DSS that can address all possible scenarios;

Table 3
Maintenance data for the 20 components of the OWT considered in the case study, along with the condition monitoring system data.

Subsystem	Component	State	Trsn params.		Required maintenance		Action
	#-Name		β	η	Type	Action	cost (£)
Blades	1-Blades	Degraded	1.2	23.02	5	Minor repair (patching, sealing)	12,000
		Critical	1.2	2.88	1	Replacement	600,000
		Failed	1.2	2.88	1	Replacement	600,000
Hub	2-Hub	Degraded	1.2	15.38	4	Minor corrosion repair	9,000
		Degraded	1.2	1.92	1	Replacement	132,000
		Failed	1.2	1.92	1	Replacement	132,000
Drivetrain	3-Main bearings	Degraded	1.2	160	4	Repair of pitting, misalignment	15,000
		Critical	1.5	20	1	Bearing replacement	60,000
		Failed	1.5	20	1	Bearing replacement	60,000
	4-Gearbox	Degraded	1.3	16	4	Gear tooth repair	15,000
		Critical	1.2	2	3	Gear replacement	150,000
		Failed	1.4	2	2	Gearbox replacement	780,000
	5-Main shafts	Degraded	1.2	160	4	Minor repair	15,000
		Critical	1.5	20	4	Minor repair, adjust alignment	15,000
		Failed	1.5	20	1	Shaft replacement	110,000
Break System	6-Calipers/pads	Degraded	1.2	13.11	3	Replace worn components	12,000
		Failed	1.2	3.28	3	Replace calipers	12,000
	7-Brake Discs	Degraded	1.2	42.28	3	Replacement	12,000
		Failed	1.2	10.57	3	Replacement	12,000
Power System	8-Generator	Degraded	1.2	15.38	3	Part replacement	150,000
		Critical	1.2	1.92	3	Part replacement	150,000
		Failed	1.2	1.92	2	Replacement	450,000
	9-Frequency converter	Degraded	1.2	33.38	2	Replacement	36,000
		Failed	1.2	8.35	2	Replacement	36,000
	10-Transformer	Failed	1.2	14.93	3	Replacement	90,000
Hydraulic System	11-gear pump	Degraded	1.2	12	3	Part replacement	78,000
		Failed	1.2	8	3	Pump replacement	117,000
	12-Valves/pipes	Degraded	1.2	13.11	4	Tightening/replacement	3,000
		Failed	1.2	3.28	4	Replacement	3,000
Yaw System	13-Hydraulic actuator	Degraded	1.2	32.28	4	Minor repair	21,000
		Critical	1.2	12	3	Part replacement	60,000
		Failed	1.2	8.57	3	Complete replacement	90,000
	14-Bearing/gear	Degraded	1.2	29.12	4	Corrective repair	15,000
		Critical	1.2	3.64	3	Gear tooth repair	21,000
		Failed	1.2	3.64	1	Complete replacement	27,000
	15-Yaw brake	Degraded	1.2	29.12	3	Part replacement	18,000
		Critical	1.2	3.64	3	Part replacement	18,000
		Failed	1.2	3.64	3	Complete replacement	27,000
Pitch System	16-Hydraulic actuator	Degraded	1.2	25.12	4	Minor repair	24,000
		Critical	1.2	4	3	Part replacement	69,000
		Failed	1.2	7.28	3	Complete replacement	103,500
	17-Bearing/gear	Degraded	1.2	15.38	4	Corrective repair	24,000
		Critical	1.2	1.92	3	Gear tooth repair	69,000
		Failed	1.2	1.92	3	Complete replacement	69,000
Structure	18-Tower	Degraded	1.2	6	5	Corrosion repair	60,000
		Critical	1.2	3	1	Replacement	792,000
		Failed	1.2	5.93	1	Replacement	792,000
	19-Nacelle	Degraded	1.2	133.33	5	Loss of section, crack repair	15,000
		Critical	1.2	16.67	5	Loss of section, crack repair	15,000
		Failed	1.2	16.67	1	Replacement	120,000
	20-Foundation	Degraded	1.2	133.33	5	Corrosion/repaint/remove-	45,000
		Critical	1.2	16.67	5	Marine growth	45,000
		Failed	1.2	16.67	1	Replacement	612,000
CMS	21-CMS	Degraded	1.2	133.33	–	Minor repair	9,000
		Critical	1.2	16.67	–	Major repair	30,000
		Failed	1.2	16.67	–	Replacement	60,000

Table 4
Description of the nodes from the *i*PN model shown in Figs. 6 to 10.

Petri net	Nodes	Description
Fig. 6	p_{i1}, p_{i2}, p_{i3}, p_{i4}	Normal, degraded, critical, and failed true states of a component
	t_{i1}, t_{i2}, t_{i3}	Degradation process of the component
	p_{i5}, p_{i6}, p_{i7}, p_{i8}	Normal, degraded, critical, and failed known states of a component
	p_1, p_2, p_3, p_4	Normal, degraded, critical, and failed true conditions of the CMS
	t_{i1}, t_{i2}, t_{i3}	Degradation process of the CMS
	p_{34}, t_{41}	Update the known condition of the CMS
	p_{30}, p_{31}, p_{32}, p_{33}	Normal, degraded, critical, and failed known conditions of the CMS
	p_{i10}	The known condition should be updated because the component's or the CMS's condition is changed

(*continued on next page*)

Table 4 (*continued*).

Petri net	Nodes	Description
	p_{i15}	The known condition is changing
	t_{i12}, t_{i5}	Cannot or can reveal known condition because CMS is in the failed or working state
	t_{i7}	CMS will reveal condition without error
	t_{i8}, t_{i9}	CMS will reveal condition with a probability of error (degraded CMS)
	t_{i9}	Known condition to be updated without error
	p_{i12}	Known condition to be updated with error
	p_5	A Component's known condition is changed, a decision has to be taken regarding the repair of each component since that the RL state is changed
	p_{u1}	Activate the Petri net responsible for updating state (PN in Fig. 7)
	t_4	Inspection triggered
	t_{32}, t_{33}, t_{34}, t_{35}	Update the known condition of the CMS, and decide on repair if the CMS is not in the normal state
	p_{28}	Decide on repairing CMS
	t_{40}, t_{36}	Repair and do not repair decisions of CMS
	t_{37}, t_{38}, t_{39}	Repair CMS according to its condition
	t_{42}	Change the known condition to failed when the true condition is failed without the need of inspection because a failed CMS can be revealed by not receiving information about the OWT
	p_{35}, t_{t43}, t_{44}	To give an order to check maintenance for all components in case the inspection is triggered while CMS is not in the normal condition
	p_{36}	Number of failures counter
	p_{37}	Indicator that the system is in a failed state. This inhibits the degradation of other components
	t_{45}	To reset the failed state indicator once all the failed components are repaired
Fig. 8	p_{25}, p_{26}	Working and stopped states of the OWT
	t_5	Check the need of maintenance for all components. p_{i15} inhibits the start of check until no component is changing its known condition
	p_{m1}	Maintenance check starts
	p_{m5}	Component is chosen to be repaired
	p_{m8}	Component is under repair
	t_{m3}	Terminate maintenance check since the state is normal
	t_{m1}	Terminate maintenance check, maintenance already chosen
	t_{m2}	Continue check maintenance
	p_{m3}	Decide on repairing the component
	t_{m7}, t_{m8}	Choose or reject repair
	t_{m9}, t_{m10}, t_{m11}	Start preparing for maintenance according to known condition by triggering Petri net shown Fig. 9
	t_8	Start maintenance after finishing preparations in Fig. 9
	t_{m13}	Maintenance was not selected for this component (it can be selected for others), cancel maintenance for it
	t_{m12}	The component is already under maintenance, cancel maintenance for it
	t_{m21}	Carry on maintenance
	p_{m6}	Activate the Petri net shown in Fig. 10 for checking if maintenance can continue based on what was the revealed condition of the component
	p_{m9}	Maintenance cannot continue
	p_{m10}	Maintenance can continue, fire t_{m16} to change the state of OWT to stopped
	t_{i3}	Change the OWT to stopped because a component is in the failed state
	t_{m17}	Cancel repair because the component's true condition is normal
	t_{m18}, t_{m19}, t_{m20}	Repair the component according to its condition and update the known condition by sending token to p_{u1}. The delay parameters of these transitions depends on the repair type required for each condition of the component, and they are provided in Table 1
Fig. 10	p_{c1}, p_{c2}, p_{c3}	The known condition when the repair decision is taken. t_{m7} reset these places, and t_{m9}, t_{m10}, t_{m11} update them (Fig. 9)
	t_{c1} to t_{c12}	Cover all the true and known (except normal) conditions possible scenarios and control if maintenance can or cannot continue
Fig. 7	t_{u5}	Reset old known condition
	t_{u1} to t_{u4}	Update the known condition rightly based on the true condition
Fig. 9	p_9 to p_{13}	Logistic preparation is taking place for repair types 5 to 1
	t_9 to t_{13}	Logistic preparation time for repair types 5 to 1 (delay parameters are in Table 1)
	p_6, t_6	To insure that the next step cannot start until finishing logistic preparations for all repair types
	p_{14} to p_{18}	Waiting for a good weather for repair types 5 to 1
	t_{14} to t_{18}	Waiting time for a good weather will be according to the most difficult needed repair type; one transition will be fired. (delay parameters are in Table 1)
	p_7	Enable start waiting for a good weather. Waiting can be suspended by emptying it
	p_8, t_8	Travel to the sight is started. (delay parameter, that is the travel time, is in Table 1)
	p_{19} to p_{23}	Counters for how many components need each of the repair types 5 to 1
	p_{m12}	Components' known state is changed. Check if it effects the repair preparation
	t_{m25}	State of the component is changed (p_{m12}), repair preparation is undergoing (p_{24}), it was decided to repair this component (p_{m5}), and component is not already under repair (p_{m8}). Then, preparation is affected
	t_{m26}	Component's state have changed (p_{m12}), but preparation is not affected because the component was not decided to be repaired (p_{m5})
	t_{m27}	Component's state have changed (p_{m12}), but preparation is not affected because their is no undergoing repair preparation (p_{24})
	t_{m28}	Component's state have changed (p_{m12}), but preparation is not affected because the component is already undergoing repair (p_{m8})
	p_{m11}	Cancel the repair preparations related to the component by: enabling one of the cancelling repair transitions t_{m22}, t_{m22}, or t_{m24}
	t_{m22}, t_{m23}, t_{m24}	They cancel the preparation according to the last known component's condition (p_{c1}, p_{c2}, or p_{c3}). Then, reduce the repair type counters (p_{19} to p_{23}) and cancel the component's repair (p_5)
	t_{27} to t_{31}	If a repair preparation counter (p_{19} to p_{23}) is emptied because of component/s changed state/s, stop logistic preparations for that repair type
	t_{19} to t_{23}	If a repair preparation counter (p_{19} to p_{23}) is emptied because of component/s changed state/s, stop taking that repair type/s into account while waiting for a good weather
	t_{26}	If all repair counters are emptied while undergoing repair preparation (p_{24}), cancel repair preparation

- further work can consider components' conditions based on continuous damage features, along with extending the proposed methodology to farms of OWTs based on heterogeneous components and subjected to varied degradation conditions;
- more research is needed to explore the use of data directly from CMS (i.e., raw data taken from installed sensors) within the *i*PN methodology, along with its reformulation to include PHM information, thus converting it as an intelligent predictive maintenance tool.

CRediT authorship contribution statement

Ali Saleh: Investigation, Software, Methodology, Formal analysis, Writing – original draft. **Manuel Chiachío:** Conceptualization, Methodology, Writing – original draft,Supervision, Project administration, Funding acquisition. **Juan Fernández Salas:** Investigation, Writing – original draft. **Athanasios Kolios:** Writing – review & editing, Resources.

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

The data used for maintenance modelling along with the code presented in this paper, are specified within the manuscript.

Acknowledgements

This paper is part of the ENHAnCE ITN project (https://www.h2020-enhanceitn.eu/) funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 859957. The authors would also like to thank the valuable set of data from Le and Andrews [31]. Finally, the first author would like to acknowledge the University of Nottingham (UK) for hosting him during part of the course of this work.

Appendix

See Tables 3 and 4.

References

[1] Mordor Intelligence. Offshore wind turbine market - growth, trends, Covid-19 impact, and forecast (2022 - 2027). Technical Report, 2022.
[2] European Commission. Wind energy: Why the EU supports wind energy research and innovation. 2021, https://ec.europa.eu/info/research-and-innovation/research-area/energy-research-and-innovation.
[3] Jaganmohan M. Worldwide capacity of offshore wind power 2009–2020. Technical Report, 2021, https://www.statista.com/statistics/476327/global-capacity-of-offshore-wind-energy/.
[4] Yeter B, Garbatov Y, Soares CG. Life-extension classification of offshore wind assets using unsupervised machine learning. Reliab Eng Syst Saf 2022;219:108229.
[5] Scheu M, Matha D, Hofmann M, Muskulus M. Maintenance strategies for large offshore wind farms. Energy Procedia 2012;24:281–8.
[6] Santos F, Teixeira ÂP, Soares CG. Modelling and simulation of the operation and maintenance of offshore wind turbines. Proc Inst Mech Eng O: J Risk Reliab 2015;229(5):385–93.
[7] Koukoura S, Scheu MN, Kolios A. Influence of extended potential-to-functional failure intervals through condition monitoring systems on offshore wind turbine availability. Reliab Eng Syst Saf 2021;208:107404.
[8] Yeter B, Garbatov Y, Soares CG. Risk-based maintenance planning of offshore wind turbine farms. Reliab Eng Syst Saf 2020;202:107062.
[9] Nilsson J, Bertling L. Maintenance management of wind power systems using condition monitoring systems—life cycle cost analysis for two case studies. IEEE Trans Energy Convers 2007;22(1):223–9.
[10] Marugán AP, Chacón AMP, Márquez FPG. Reliability analysis of detecting false alarms that employ neural networks: A real case study on wind turbines. Reliab Eng Syst Saf 2019;191:106574.
[11] Besnard F, Fischer K, Tjernberg LB. A model for the optimization of the maintenance support organization for offshore wind farms. IEEE Trans Sustain Energy 2013;4(2):443–50. http://dx.doi.org/10.1109/TSTE.2012.2225454.
[12] Yürüşen NY, Rowley PN, Watson SJ, Melero JJ. Automated wind turbine maintenance scheduling. Reliab Eng Syst Saf 2020;200:106965.
[13] Ghamlouch H, Fouladirad M, Grall A. The use of real option in condition-based maintenance scheduling for wind turbines with production and deterioration uncertainties. Reliab Eng Syst Saf 2019;188:614–23. http://dx.doi.org/10.1016/j.ress.2017.10.001.
[14] Hofmann M, Sperstad IB. NOWIcob–A tool for reducing the maintenance costs of offshore wind farms. Energy Procedia 2013;35:177–86.
[15] Sahnoun M, Baudry D, Mustafee N, Louis A, Smart PA, Godsiff P, Mazari B. Modelling and simulation of operation and maintenance strategy for offshore wind farms based on multi-agent system. J Intell Manuf 2019;30(8):2981–97.
[16] Obdam T, Rademakers L, Braam H, Eecen P. Estimating costs of operation & maintenance for offshore wind farms. In: Proceedings of European offshore wind energy conference. 2007, p. 4–6.
[17] Nguyen TAT, Chou S-Y. Maintenance strategy selection for improving cost-effectiveness of offshore wind systems. Energy Convers Manage 2018;157:86–95.
[18] Abdollahzadeh H, Atashgar K, Abbasi M. Multi-objective opportunistic maintenance optimization of a wind farm considering limited number of maintenance groups. Renew Energy 2016;88:247–61.
[19] Sarker BR, Faiz TI. Minimizing maintenance cost for offshore wind turbines following multi-level opportunistic preventive strategy. Renew Energy 2016;85:104–13.
[20] Pérez E, Ntaimo L, Byon E, Ding Y. A stochastic DEVS wind turbine component model for wind farm simulation. In: Proceedings of the 2010 spring simulation multiconference. 2010, p. 1–8.
[21] Byon E, Pérez E, Ding Y, Ntaimo L. Simulation of wind farm operations and maintenance using discrete event system specification. Simulation 2011;87(12):1093–117.
[22] Jagtap HP, Bewoor AK, Kumar R, Ahmadi MH, Chen L. Performance analysis and availability optimization to improve maintenance schedule for the turbo-generator subsystem of a thermal power plant using particle swarm optimization. Reliab Eng Syst Saf 2020;204:107130.
[23] Wang J, Miao Y. Optimal preventive maintenance policy of the balanced system under the semi-Markov model. Reliab Eng Syst Saf 2021;213:107690.
[24] Endrerud O-EV, Liyanage JP. Decision support for operations and maintenance of offshore wind parks. In: Engineering asset management-systems, professional practices and certification. Springer; 2015, p. 1125–39.
[25] Endrerud O-EV, Liyanage JP, Keseric N. Marine logistics decision support for operation and maintenance of offshore wind parks with a multi method simulation model. In: Proceedings of the winter simulation conference 2014. IEEE; 2014, p. 1712–22.
[26] Stock-Williams C, Swamy SK. Automated daily maintenance planning for offshore wind farms. Renew Energy 2019;133:1393–403.
[27] Compare M, Martini F, Zio E. Genetic algorithms for condition-based maintenance optimization under uncertainty. European J Oper Res 2015;244(2):611–23.
[28] Izquierdo J, Márquez AC, Uribetxebarria J. Dynamic artificial neural network-based reliability considering operational context of assets. Reliab Eng Syst Saf 2019;188:483–93.
[29] Leigh JM, Dunnett SJ. Use of Petri nets to model the maintenance of wind turbines. Qual Reliab Eng Int 2016;32(1):167–80.
[30] Yan R, Dunnett S. Improving the strategy of maintaining offshore wind turbines through Petri net modelling. Appl Sci 2021;11(2):574.
[31] Le B, Andrews J. Modelling wind turbine degradation and maintenance. Wind Energy 2016;19(4):571–91.
[32] Andrews J, Fecarotti C. System design and maintenance modelling for safety in extended life operation. Reliab Eng Syst Saf 2017;163:95–108.
[33] Chiachío M, Saleh A, Naybour S, Chiachío J, Andrews J. Reduction of Petri net maintenance modeling complexity via approximate Bayesian computation. Reliab Eng Syst Saf 2022;222:108365.
[34] Murata T. Petri nets: Properties, analysis and applications. Proceesings of the IEEE 1989;77:541–80.
[35] Zurawski R, Zhou M. Petri nets and industrial applications: A tutorial. IEEE Trans Ind Electron 1994;41(6):567–83.
[36] Pei-Ming B. Learning capability in fuzzy Petri nets based on BP net. Chinese J Comput 2004;5.
[37] Hanna MM, Buck A, Smith R. Fuzzy Petri nets with neural networks to model products quality from a CNC-milling machining centre. IEEE Trans Syst Man Cybern- A: Syst Hum 1996;26(5):638–45.
[38] Lee J, Liu K, Chiang W. Modeling uncertainty reasoning with possibilistic Petri nets. IEEE Trans Syst Man Cybern B 2003;33(2):214–24.
[39] Chiachío M, Chiachío J, Prescott D, Andrews J. A new paradigm for uncertain knowledge representation by plausible Petri nets. Inform Sci 2018;453:323–45.
[40] Chiachío M, Chiachío J, Prescott D, Andrews J. Plausible Petri nets as self-adaptive expert systems: A tool for infrastructure asset monitoring. Comput-Aided Civ Infrastruct Eng 2019;34(4):281–98.

[41] Vidal JC, Lama M, Bugarín A. Petri net-based engine for adaptive learning. Expert Syst Appl 2012;39(17):12799–813.

[42] Serral E, De Smedt J, Snoeck M, Vanthienen J. Context-adaptive Petri nets: Supporting adaptation for the execution context. Expert Syst Appl 2015;42(23):9307–17.

[43] Sutton RS, Barto AG. Reinforcement learning: an introduction. MIT Press; 2018.

[44] Pinciroli L, Baraldi P, Ballabio G, Compare M, Zio E. Optimization of the Operation and Maintenance of renewable energy systems by Deep Reinforcement Learning. Renew Energy 2022;183:752–63.

[45] Fan L, Su H, Wang W, Zio E, Zhang L, Yang Z, Peng S, Yu W, Zuo L, Zhang J. A systematic method for the optimization of gas supply reliability in natural gas pipeline network based on Bayesian networks and deep reinforcement learning. Reliab Eng Syst Saf 2022;225:108613.

[46] Lee J, Mitici M. Deep reinforcement learning for predictive aircraft maintenance using Probabilistic Remaining-Useful-Life prognostics. Reliab Eng Syst Saf 2022;108908.

[47] Mattila V, Virtanen K. Scheduling fighter aircraft maintenance with reinforcement learning. In: Proceedings of the 2011 winter simulation conference (WSC). IEEE; 2011, p. 2535–46.

[48] Yang H, Li W, Wang B. Joint optimization of preventive maintenance and production scheduling for multi-state production systems based on reinforcement learning. Reliab Eng Syst Saf 2021;214:107713.

[49] Nguyen V-T, 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.

[50] Mohammadi R, He Q. A deep reinforcement learning approach for rail renewal and maintenance planning. Reliab Eng Syst Saf 2022;108615.

[51] 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.

[52] Kuhnle A, Jakubik J, Lanza G. Reinforcement learning for opportunistic maintenance optimization. Prod Eng 2019;13(1):33–41.

[53] Chatterjee J, Dethlefs N. Deep reinforcement learning for maintenance planning of offshore vessel transfer. In: Developments in renewable energies offshore. CRC Press; 2020, p. 435–43.

[54] Sierra-García JE, Santos M. Exploring reward strategies for wind turbine pitch control by reinforcement learning. Appl Sci 2020;10(21). http://dx.doi.org/10.3390/app10217462.

[55] Sedighizadeh M, Rezazadeh A. Adaptive PID controller based on reinforcement learning for wind turbine control. In: Proceedings of world academy of science, engineering and technology, Vol. 27. Citeseer; 2008, p. 257–62.

[56] Dong W, Zhao T, Wu Y. Deep reinforcement learning based preventive maintenance for wind turbines. In: 2021 IEEE 5th conference on energy internet and energy system integration (EI2). IEEE; 2021, p. 2860–5.

[57] Koch M, Rust C, Kleinjohann B. Design of intelligent mechatronical systems with high-level Petri nets. In: Proceedings 2003 IEEE/ASME international conference on advanced intelligent mechatronics (AIM 2003), Vol. 1. IEEE; 2003, p. 217–22.

[58] Lee J-H, Kim H-J. Reinforcement learning for robotic flow shop scheduling with processing time variations. Int J Prod Res 2021;1–23.

[59] Drakaki M, Tzionas P. Manufacturing scheduling using colored Petri nets and reinforcement learning. Appl Sci 2017;7(2):136.

[60] Hu L, Liu Z, Hu W, Wang Y, Tan J, Wu F. Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. J Manuf Syst 2020;55:1–14.

[61] Watkins CJ, Dayan P. Q-learning. Mach Learn 1992;8(3):279–92.

[62] Jensen K, Rozenberg G. High-Level Petri nets: theory and application. Springer Science & Business Media; 2012.

[63] Andrews J. A modelling approach to railway track asset management. Proc Inst Mech Eng F: J Rail Rapid Transit 2013;227:56–73.

[64] Buşoniu L, Babuška R, Schutter BD. Multi-agent reinforcement learning: An overview. Innov Multi-Agent Syst Appl-1 2010;183–221.

[65] Busoniu L, Babuska R, De Schutter B. A comprehensive survey of multi-agent reinforcement learning. IEEE Trans Syst Man Cybern C (Appl Rev) 2008;38(2):156–72.

[66] Kang J, Soares CG. An opportunistic maintenance policy for offshore wind farms. Ocean Eng 2020;216:108075.

[67] IRENA-international renewable energy agency. 2020, https://www.irena.org/wind, Accessed: 2022-02-13.

[68] Offshore wind outlook 2019. 2022, https://www.iea.org/reports/offshore-wind-outlook-2019, Accessed: 2022-02-13.

[69] United Kingdom electricity prices. 2021, https://www.globalpetrolprices.com/United-Kingdom/electricity_prices/, ????, Accessed: 2022-08-25.

[70] How much does wind turbine cost worth it in 2022. 2021, https://weatherguardwind.com/how-much-does-wind-turbine-cost-worth-it/, Accessed: 2022-06-12.