An Assessment of Different Reinforcement Learning Methods for Creating a Decision Support System Based on the Petri Net Model

Ali Saleh[1], Manuel Chiachio[2]

[1,2] *Andalusian Research Institute in Data Science and Computational Intelligence, University of Granada, 18071 Granada, Spain*
alisaleh@ugr.es
mchiachio@ugr.es

ABSTRACT

Infrastructure safety, operation, and management are directly affected by operation strategy which directly influence the life cycle and have a tremendous impact on operation costs. Management strategies can be branched to unrelated decisions which are typically difficult to be formulated mathematically. Reinforcement Learning serves as an adequate tool to account for unrelated decisions and optimize them in accordance with a final goal. Besides, it allows for a strategy to be flexible by autonomously changing with the variation of the system condition without the need for any user intervention. On the other hand, Petri nets are a suitable tool for maintenance modeling, as they can deal with heterogeneous information, parallel operations, and synchronization, and provide a graphical interpretation for the processes. Also, they allow for formulating RL problems based on their standard components (namely, places and transitions) which makes them suitable for forming a hybrid tool with the RL. In this work, an intelligent tool for modeling and optimizing maintenance has been developed by combining high-level Petri net (HLPN) and Reinforcement Learning (RL). RL actions are modeled by creating a special type of PN transitions, and rewards are given in terms of the running revenues and costs. The method was used to model a safety-critical system (SCS) and optimize its maintenance schedule in order to avoid the catastrophic consequences of failures while reducing the running costs. The results show that the proposed approach is capable of providing an intelligent management tool for SCS operation under flexible, yet unveiled policies, and also allows assessment of the safety of the system based on the simulated system states.

1. INTRODUCTION

The assessment of the system's safety, reliability, and risk are important tasks performed through the life-cycle of any system to avoid undesired consequences. These tasks have special importance in safety-critical systems (SCS) because of the severe harm or damage that can be caused to people, the environment, or equipment/property in case of damage or malfunction. These systems are spread out now a day in many industries including automotive, medical, energy, aerospace, nuclear, and process industries which make them an important part of everyday life in modern society. For this, analysts have to understand the behavior of such systems to reach high dependability levels; which means understanding how they work to avoid failures that are more severe and/or frequent.

Several classical approaches have been used over the years for doing this job like Fault Tree Analysis (FTA) (Stamatelatos et al., 2002), Bowtie diagrams (De Dianous & Fiévez, 2006), Hazard and Operability Analysis (HAZOP) (Dunjó, Fthenakis, Vílchez, & Arnaldos, 2010), Failure Modes Effect and Criticality Analysis (FMECA) (Standard, 1980). However, these approaches require making unrealistic assumptions such as binary states, statistically independent, and unrepairable components, a single mode of operation for the system, availability of failure data, and focusing only on the technical part of the system. Some work was done to improve the classical approaches like extending the FTA to dynamic fault tree analysis (DFT) (Dugan, Bavuso, & Boyd, 1992) or to Temporal Fault Trees (TFTs) (Palshikar, 2002). Moreover, Markov chains, PNs, and simulations (e.g. Monte Carlo) have been applied to overcome the classical approaches in this field. Markov chains are limited to systems with exponential distributions lifetime and suffer from state-space explosion (Kabir & Papadopoulos, 2019). On the other hand, simulations can work with different kinds of distributions and can be used when it is difficult to analyze state-space, but it requires more memory (Kabir & Papadopoulos, 2019). PNs can overcome the limitations of the aforementioned models by being applicable for concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic systems while combining graphical representation of the system's dynamics with a well-defined mathematical theory. They also provide a one-to-one interface that cannot be handled with other for-

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

malisms, such as formal specification and verification(Kabir & Papadopoulos, 2019). This made them widely applied to system safety, reliability, and risk assessment domain (N. G. Leveson & Stolzy, 1987; N. Leveson, Dulac, Marais, & Carroll, 2009; Kabir & Papadopoulos, 2019).

One of the main goals of SCS is trying to minimize the likelihood of potential risks after identifying them. This requires knowing the optimal actions at each of the system's states that lead to reducing the risk. PNs are able to perform stochastic simulations to reveal all the possible states of a system and assess its reliability under one or more policies. However, complex systems can result in a huge number of available strategies and states, which requires intelligent tools to explore important areas of the search space. Reinforcement learning (RL) is an adequate tool for finding an optimal policy, where a policy is an identification of the actions taken at each of the system's states. This can be done by interacting with the system's environment by performing number of PN simulations after defining the goal of the problem. RL is a group of machine learning methods that rely on the concept of teaching an agent from experience, trying to mimic the nature of organisms (Sutton & Barto, 2018). RL can help PN not only identify the possible existing scenarios and assess risk but also find the optimal behavior that minimizes risk and increases the system's safety. This works by giving rewards or punishments as positive or negative values to an agent experiencing the environment as a consequence of its chosen actions. Then, evaluate a value function (section 2.1) for taking an action at a given state based on the sequence of the resulted rewards after that action.

RL was used with PN model for few applications including manufacturing scheduling (Drakaki & Tzionas, 2017) and the designing mechatronical systems with adaptive behavior (Koch, Rust, & Kleinjohann, 2003). In this paper, the use of RL with PN model is extended to create an expert decision support system (DSS) that performs reliability analysis, namely, for modeling an SCS and optimizing its maintenance schedule to reach high dependability levels. Section 2 gives a brief overview of the RL, PN model, and the way the two methods are combined. Then, Section 3 describes a case study of an SCS that is modeled and optimized using the proposed method. Results and discussion on the outcome of the described case are shown Section 4. Then, concluding remarks are provided in Section 5.

2. METHODOLOGY

2.1. Reinforcement learning

RL is a way to teach machines from interaction with the environment. The main elements of RL are the agent, environment, reward, value function, and policy. The agent is the learning element that interacts with the environment. The learning process happens when the environment gives positive and negative rewards for the acts the agent is doing. The goal of these rewards is to help the agent evaluate how good it is to take an an action at a given state. This evaluation is known as the value function of the state which is a representation of long-term rewards coming after it. It is needed to evaluate the values to teach the agent an optimal policy that can increase its long-term rewards. One way to formalize the RL problem is by using Markov Decision Processes (MDPs) which assume that the probability distribution of future states is only a function of the current state and action. At each time step, the agent take an action, A_t from a state S_t; then the environment is changed to state S_{t+1} and gives a reward R_{t+1}. The sequence of rewards given after a state is known as the expected return:

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

where $\gamma = [0, 1]$ is the discount rate parameter with 1 meaning no discount. RL problems can be continuous or episodic, and they are distinguished by that the episodic task has a terminating state (end) while the continuous task can continue without an end. The learning process in episodic tasks is formed of several episodes, with each episode identified by its terminating state. γ can't be specified to be equal to 1 in continuous tasks, otherwise, the summation of future rewards can diverge to $G_t = \pm\infty$ (Sutton & Barto, 2018). Q-learning can be used to calculate the value function of each state-action pair according to the following update equation (Watkins & Dayan, 1992):

$$\begin{aligned} Q(S_t, A_t) = & Q(S_t, A_t) + \alpha[R_{t+1} + \\ & \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \end{aligned} \quad (2)$$

where α is called the learning rate parameter. This parameter should be always between zero and one, and it represents how much it is required to keep knowledge from previous updates when updating the value function. If it is equal to 0 then no learning will occur after experiencing the environment, and if it is equal to 1 then no knowledge will be kept from old experiences. Q-learning use bootstrapping which is truncating the summation terms of Equation 1 at the second term and using the value function of the next state instead of G_{t+1}. This enables calculating the expected return G_t before the end of the episode which makes online updating of the value function possible. Besides, Equation 2 enables updating $Q(S_t, A_t)$ as an expected value of calculated returns without the need to save all them all.

Q-learning updates G_t based on the value of the action that has the highest return as shown in the equation. Now, the policy improvement step can be implemented by acting greedy with respect to the Q-values which means choosing the action that has the highest Q-value in each state. However, the

knowledge at the beginning of the problem is not reliable, and choosing the decision that seems to be optimal can prevent updating other actions that may be better than the chosen one. This makes exploration and exploitation both essential for the process of reaching an optimal policy, but they can't also happen at the same time. This is known as the exploration-exploitation dilemma that exists in almost all RL methods. To solve this, an ε-greedy strategy can be followed. This strategy keeps a probability of ε for exploration each time a decision is taken:

$$A_t = \begin{cases} \arg\max_a Q(S,a), & \text{with probability (1-}\varepsilon\text{)} \\ A \in_R \mathcal{A}(s), & \text{with probability } \varepsilon \end{cases} \tag{3}$$

where ε is the exploration rate parameter that should be specified between 0 and 1. Exploration is maximum when ε is equal to 1 and minimum when it is equal to 0. This approach ensures that each Q-value will be updated infinite times as the number of iterations approaches infinity which ensures the convergence of Q-values while exploiting the knowledge. At the beginning of the problem, there is no knowledge to exploit, so ε can start with 1 and decrease gradually until the end of the problem when exploitation becomes more important.

2.2. Petri net

A PN is a directed bipartite composed of two types of nodes known as places and transitions. The places describe the state of the PN and they are depicted by circles while transitions are responsible for describing the changes in the system's state and are depicted by rectangles. Each place has a *marking* that is the number of tokens in that place; where tokens are the abstract moving units of a PN and depicted by black dots. The state of a PN is then described by a vector formed from the markings of all places. Places and transitions are connected by weighted arcs with a default weight equal to 1. The weights of the arcs determine the number of tokens to be generated or dissipated in the *pre-set* or *post-set* places of a transition after it fires.

Mathematically, a PN is defined as a tuple $\mathfrak{N} = \langle \mathbf{P}, \mathbf{T}, \mathbf{F}, \mathbf{W}, \mathbf{M}_0 \rangle$, where $\mathbf{P} \in \mathbb{N}^{n_p}$, $\mathbf{T} \in \mathbb{N}^{n_t}$, $\mathbf{F} \subseteq (\mathbf{P} \times \mathbf{T}) \cup (\mathbf{T} \times \mathbf{P})$ represent the set of arcs connecting places and transitions, $\mathbf{W} : \mathbf{F} \to \mathbb{N}_{>0}$ represent the set of weights of the arcs, and $\mathbf{M}_0 : \mathbf{P} \to \mathbb{N}$ is the initial marking of the PN (Murata, 1989) . The dynamics of a PN are controlled by the *firing rule* which determines when a transition can fire, and what are the consequences of firing a transition (David & Alla, 2010). This rule, in its simplest form, says that if the number of tokens in the pre-set places of a transition is greater than or equal to the weights of its pre-set arcs, the transition can fire.

However, to account for the complexity of practical applications, the definition of Timed Petri net (TPN) is used which gives delay time for the transitions before they fire. Transitions with delays can describe processes that require time to happen like degradation of a component. If the delay of transitions is based on stochastic distribution, the PN is called Stochastic Petri net (SPN) which is the type of model created in this paper. Also, the inhibitor and reset arcs that are special types of arcs represented by unfilled and filled circular tips respectively as shown in Figure 2 are used. The inhibitor arc makes the opposite effect of the normal arc by disabling the transition once a token exists in any pre-set place of the transition. Whereas, the reset arc changes the marking of a post-set place of a transition once it fires to a value defined by the user. This value can be defined to be the initial marking of the place, zero marking, or any other number. In this paper, all the reset arcs are defined to change the markings of the connected post-set places to their initial values.

After the firing of a transition, number of tokens equal to the weights of the pre-set arcs will be consumed from pre-set places, and number of tokens equal to the weights of post-set arcs will be generated of the post-set places. Accordingly, the firing of transitions causes the change in the markings of the places, and this change can be described mathematically using the state equation defined as:

$$\mathbf{M}_{k+1} = \mathbf{M}_k + \mathbf{A}^T \mathbf{u}_k \tag{4}$$

where $\mathbf{M}_k$ is the marking vector at time step k and $\mathbf{u}_k = (u_{1,k}, u_{2,k}, \ldots, u_{n_t,k})^T$ is the *firing vector*. $\mathbf{A}$ is an $n_t \times n_p$ matrix representing the difference between weights of input and output arcs connecting places and transitions. This matrix is referred to as the *incidence matrix*, and is calculated as follows:

$$\mathbf{A} = \mathbf{A}^+ - \mathbf{A}^- \tag{5}$$

where $\mathbf{A}^+ = \left[a_{ij}^+\right]$ and $\mathbf{A}^- = \left[a_{ij}^-\right]$, $i = 1, \ldots, n_t$, $j = 1, \ldots, n_p$ are the *incidence matrix* and *backward incidence matrix* respectively. The elements of the arrays a_{ij}^+ and a_{ij}^- coincides with the weights of arcs w_{ij}^+ and w_{ij}^- connecting transition $t_i \in \mathbf{T}$ to place $p_j \in \mathbf{P}$ respectively.

2.3. The use of Reinforcement learning with Petri net model

RL is used as a teaching method for choosing between possible actions. PN transitions are used to define RL actions by introducing a new finite set $\mathbf{G} = \{g_1, g_2 \ldots, g_{n_g}\}$, named *action groups*. Each action group contains a number of transitions that represent possible choices at a certain state. These transitions should be able to satisfy the enabling conditions at the same state for them to be choices of the same action group. Once a transition in an action group satisfies the conditions to be enabled, the action group is enabled and not the transition. Then, the RL agent chose one of the transitions in that group to be enabled based on the policy. The action group is kept enabled until the chosen transition is fired to avoid choosing another action before finishing the first one.

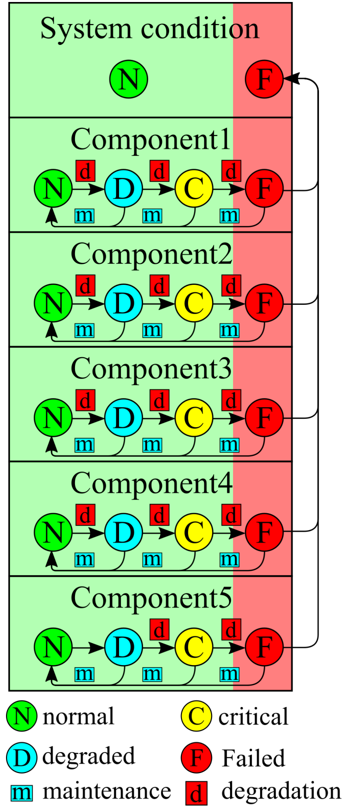

Figure 1. Illustration of the CSS showing the effect of degradation and maintenance on the state of each component and the condition of the system

It is important to distinguish between the state of the RL environment and the state of the PN. The RL environment can be defined based on any information available that may affect the choice of the agent. In the case of the PN model, the environment can include part or all of the places' markings, the status of the transitions whether they are disabled, enabled, or fired, the simulation passed time, or any other information. Thus, the two types of states are called the RL state and PN state to differentiate between them, and the RL agent takes decision based on the RL state. Defining the RL environment separately from the PN markings can have a great advantage in reducing the number of possible RL states which can significantly reduce the computational cost because each state needs to be experienced a lot of times to converge to its accurate value function. A PN with 15 binary places can have 1'048'576 different states. If the markings of only 5 places are important for RL decisions, the RL environment can be reduced to 5 places which results in 32 states.

3. Safety critical system intelligent Petri net model

A PN model is created to simulate the degradation and maintenance, and optimize the maintenance strategy to form a reliable DSS for any system of any number of components. The model is used to simulate the case of a SCS that its failure can cause severe damage to the equipment with significant losses. The values used to describe the different processes of the system are based on assumptions, but the same model can be used for any other real case example by only changing the transitions and rewards values. The simulated system is illustrated in Figure 1, which is formed of 5 components and the failure of any of them can cause the system's failure. Each of the components can be in a normal, degraded, critical, and failed states. It is assumed that the components are condition monitored with no errors, and the maintenance actions are taken based on the component's revealed condition. Thus, the only followed maintenance technique is condition based maintenance.

The PN shown in Figure 2 represent part of the model, with the nodes in the shaded area is to model component 1, and this is repeated for all other components, while other nodes is to model the system common parts. The hanging arcs represent the connections to the other unshown components. The nodes' names for each component is followed by the component index, so for the component presented, the names are followed by the index '-1'. Places p_1, p_2, and p_3 that are outside this area are to model system's functions and they represent the working system state, the non-working system state, and the number of system's failures respectively. For a component of index n, the marking of place p_{1-n} represents the state of the component. A marking equals to 3, 2, 1, or 0 represents the normal, degraded, critical, or failed states. Once the marking reaches 0, t_{6-n} fires to add a token to the system's number of failures counter represented by p_3. Transition t_{1-n} represents the degradation processes of the component. Since the degradation transition from a state to the following is function of the current state, the firing delay of the transition t_{1-n} is linked to the marking of place p_{1-n}. Thus, every time t_{1-n} is enabled, a delay will be chosen based on the marking of p_{1-n}. The values of delay for t_{1-n} at each state of each component are presented as transition functions based on probability distributions in Table 1.

The firing of t_{1-n} changes the state of the component which changes the state of the system. At the same time, the decision to repair a component should be done based on the overall state of the system that represent the RL environment, not on the individual state of the component. Thus, it is required to take a decision for every component every time any component's state changes. For this, the firing of t_{1-n} marks places $p_{2-i} \forall i = 1 \cdots n_c$, where n_c is the total number of components. Then, if component n is in the normal state, t_{3-n} fires to dissipate that token since no need for making any decision. Whereas, if it is in a deteriorated state, t_{2-n} fires to mark p_2 that stop the system functioning and consider the downtime of the system, and to mark p_{3-n} that activates action group g_n to make a decision regarding the repair of the component.

Component #	State	Transition function (years)	Maintenance cost $	Maintenance time (days)
1	Degraded	$\mathcal{W}(1.2, 3.4)$	1'200	1
	Critical	$\mathcal{W}(1.2, 8.2)$	1'700	2
	Failed	$\mathcal{W}(1.2, 5.1)$	1'300'000	10
2	Degraded	$\mathcal{W}(1.2, 3.3)$	900	1.5
	Critical	$\mathcal{W}(1.2, 7.8)$	1'400	2.5
	Failed	$\mathcal{W}(1.2, 4.0)$	2'000'000	10
3	Degraded	$\mathcal{W}(1.2, 3.5)$	2'000	2
	Critical	$\mathcal{W}(1.2, 7.5)$	3'000	2.5
	Failed	$\mathcal{W}(1.2, 5.0)$	1'200'000	10
4	Degraded	$\mathcal{W}(1.2, 3.2)$	1'300	0.5
	Critical	$\mathcal{W}(1.2, 8.4)$	1'800	2
	Failed	$\mathcal{W}(1.2, 2.0)$	1'600'000	10
5	Degraded	$\mathcal{W}(1.2, 3.4)$	1'000	1.5
	Critical	$\mathcal{W}(1.2, 8.3)$	1'500	3
	Failed	$\mathcal{W}(1.2, 5.0)$	1'000'000	10

Table 1. Transition function, maintenance cost, and maintenance time for each component at each state. $\mathcal{W}(\beta, \eta)$ is Weibull distribution with shape β and scale η.

The action group g_n includes the transitions t_{4-n} and t_{5-n} that represent the repair and don't repair decisions respectively. Transition t_{4-n} is a delayed transition with a delay function related to the marking of place p_{1-n} to represent the time of different repair types required at each state of the component. These delays at each of the components' states are presented as maintenance time in Table 1.

The rewards of the RL are represented in monetary terms for the considered case. These rewards include the losses of the system in case it is in a failed state, and the costs of performing maintenance for each component at each of its state. The losses of the system are assumed to have a rate equal to $ 5'000 per year when the system in a failed state. On the other hand, the maintenance costs are summarized in Table 1 for each component at each of its deteriorated states. It should be noted that the maintenance costs at the failed state of each component is much greater than the other two states, and this to represent the catastrophic consequences occurring for the equipment if the system fails. An additional $ 3'000 that represent the rent of the equipment needed to perform maintenance is required each time a maintenance is performed. This rent can be common for more than one component if they undergo maintenance at the same time, and this type of maintenance is known as opportunistic maintenance. The losses are considered by giving a negative rewards with the mentioned rate as long as p_2 is marked, and the costs are considered by giving a negative reward as soon as the maintenance transition t_{5-1} fires. To model the different maintenance costs as shown in Table 1 through only one transition for each component, the reward is linked to the state of the component by assigning a reward function of the marking of p_{1-n}. It is assumed that the maintenance of a component returns its state to its pristine condition. It is assumed that the lifespan of the system is 80 years, so the learning process is episodic task. The total number of episodes is chosen to be equal to 100'000 with the length of each episode equal to the lifespan of the system.

Figure 2. Maintenance actions Petri net, with shaded part of the model represent the nodes of one specific component, and other nodes are common for all components

As mentioned in Section 2.1, the used RL method requires choosing three parameters that are the discount rate, γ, the learning rate, α, and the exploration rate, ε. Since that the delayed and current rewards are of the same importance in the considered problem, it is preferable to calculate the expected return without discounting the delayed rewards in order for the agent not to be biased toward current ones. For this, the discount rate was chosen to be equal to 1, and to ensure the convergence, the value function of the terminating state of each episode was always 0 (Sutton & Barto, 2018). On the other hand, α and ε were chosen to decay gradually starting by 0.1 and 1 and ending by 0.005 and 0.001 at the end of the learning process respectively.

4. Results and discussion

This section shows the results of the simulation performed for the 5 components SCS, and a discussion about these results.

The results of the learning process (100'000 episodes) are divided into intervals of 500 episodes each to calculate the average results and uncertainty bounds within each interval. This is because the learning process is highly stochastic due to the natural stochasticity of the problem and the high exploration rate at the beginning of the learning process, and this makes it inconvenient to plot the result of each episode. Exploration is required at the beginning of the problem to discover the environment. Besides, the agent doesn't have information about the environment to exploit at that point. Whereas, the agent should be able to reach the optimal policy before reaching the end of the learning process, and this allows exploiting the learnt knowledge. This explains the choice of the decay of ε between 1 and 0.001. Because of the stochasticity, it is essential to have a low learning rate in order to keep previous knowledge learnt about the environment. However, previous knowledge at the beginning of the learning process is useless because it depends on a random policy. Also because the value functions are updated by bootstrapping that uses the values of other unconverged results. This is why α was chosen to decay gradually so that the weight of previous knowledge increases as the solution converges.

Figure 3 shows the evolution of different results as a function of the learning process. It can be noted from Figure 3a that the average accumulated rewards have increased successfully by 86.3% and that its uncertainty is highly narrowed. These results indicate that the agent was able to learn how to act optimally concerning the user's input, but it doesn't show the effect on the real-case problem. For example, if the rewards were assigned in the wrong way, the agent can still increase the rewards by learning according to the wrong assigned inputs, but it doesn't mean that the agent will be able to optimize the real problem. For this, Figures 3b, 3c, and 3d that show the average number of failures, average losses, and average maintenance costs respectively are presented to show the real results that the agent was able to learn.

The agent was able to reduce the number of failures to almost zero, decrease the losses to almost zero, and optimize the maintenance costs to decrease on average while staying greater than zero. The number of failures was reduced because the consequences of failure in terms of maintenance costs are catastrophic, and every component should be repaired after failure to avoid continuous losses that can be great if significant time passed without repair. This drove the agent to strictly avoid any failure, and the only failures that exist at the end of the problem are because of the ε-greedy strategy that allows some exploration to keep learning. The result of the reduced number of failures affected not only the maintenance costs but also the downtime of the system since the time required to repair a failed component is much greater than the that required to repair a component in any other state as shown in Table 1. The agent was able to find a policy that reduces the downtime to almost zero and this is reflected in the losses results that reached an average value very close to zero as shown in Figure 3c. Lastly, the maintenance costs were optimized by making a trade-off between costs and losses. It can be noted from the uncertainty bounds at the beginning of the process that there are cases where the maintenance costs are equal to zero, but these cases strictly don't exist by the end of the learning process. The reason behind avoiding a zero maintenance costs policy is that this policy follows the concept of work until failure while not repairing which can result in zero maintenance costs but very high losses due to the failed system. For this, the maintenance costs are optimized and decreased on average while never being equal to zero.

The final policy is represented in Figure 4 to demonstrate the results obtained in Figure 3. This figure shows the 36 states that occurred more than 450 times with their corresponding actions when following the final policy without any exploration for a simulation of 1000 episodes. The numbers listed horizontally are indexes given to the states to ease referring to them. The upper part of the figure describes the states of the RL environment that are defined as combinations of the component's states. Whereas, the lower part describes the actions corresponding to each state. For example, state number 16 is defined by the states critical, normal, degraded, normal, and normal for components 1 to 5 respectively. At this state, 4 actions as combinations of decisions for component 1 and 3 are available and they are: (repair,repair), (repair, don't repair), (don't repair,repair), and (don't repair,don't repair) with their corresponding Q-Values equal to [-2.06e5, -2.31e5, -3.11e5, and -2.91e5]. The highest Q-Value among the 4 is the first one that corresponds to the action of repairing both components, and this is why the chosen action by the agent is: repair, no action, repair, no action, no action, and no action for components 1 to 5.

The reason behind choosing to repair component 3 at state 16 although it is still in a degraded state is that the agent learnt to do opportunistic maintenance and save the common costs and time when more than one components are repaired at the same time. It was found in the final policy figure that there are no failed state for any of the components, and this compatible with the results shown in Figure 3b. Also, it was found that for the majority of cases it was better to wait until the component reaches a critical state to repair it although the maintenance costs at the critical state are higher than that at the degraded state. The explanation of this is that the time required for the component to reach a critical state is much longer than that needed to reach a degraded state which makes the maintenance costs per passed time less. For example, the first component reaches the degraded state after around 3.4 years on average while reaches the critical state after around 11.6 years on average. Maintaining this component at the degraded state after each 3.4 years can result in costs equal to $ 4080 in 11.6 years. Whereas, doing this at a critical state once

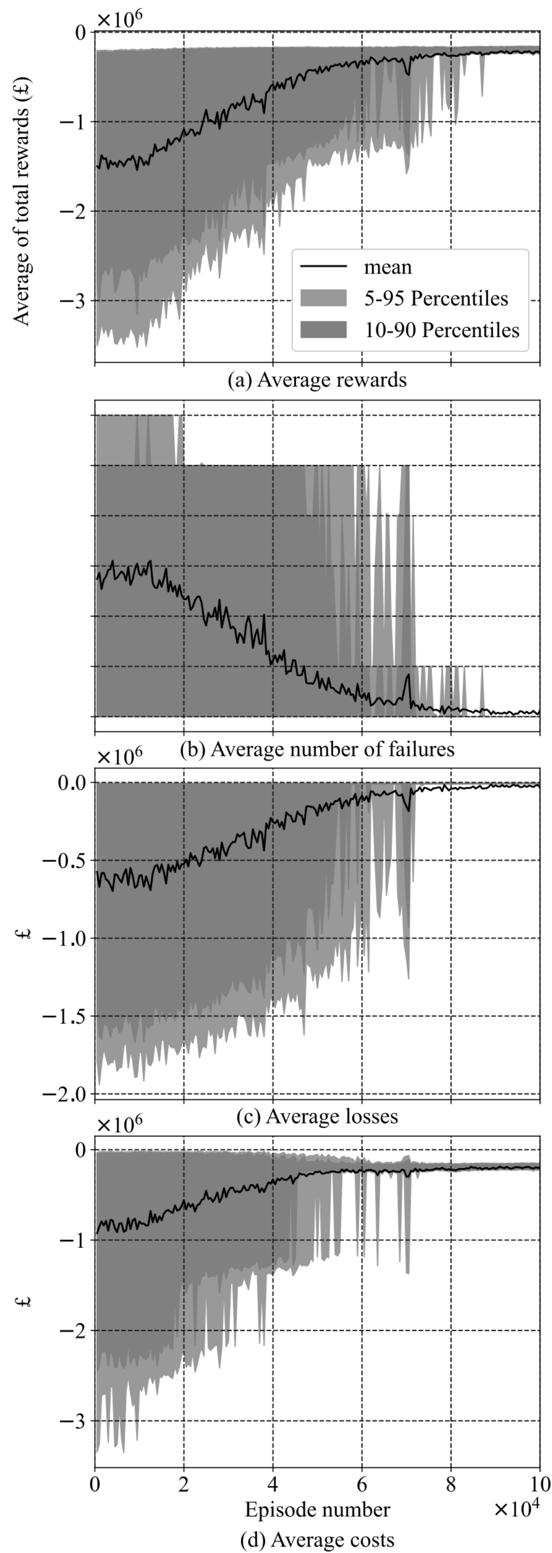

Figure 3. The evolution of different results during the RL process

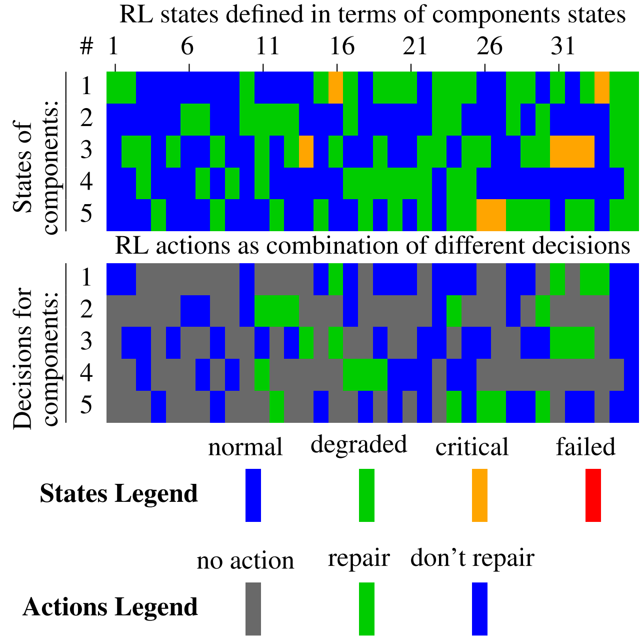

Figure 4. The most repeated RL states when following the optimal policy and the corresponding decisions at each state can result in costs equal to $ 1400 which is much cheaper.

The use of Q-learning was successful in optimizing the maintenance schedule of a SCS and avoiding the catastrophic consequences. However, for more complex problems, the method can be improved if *n-step bootstrapping* is used instead of the *one-step bootstrapping*. The *one-step bootstrapping* is good when major changes occurs between the states and this is the case of the considered problem. However, *n-step bootstrapping* allows truncating the rewards after n steps and this works better when changes are minor between consecutive states (Sutton & Barto, 2018). Also, *n-step bootstrapping* forms a general method that unify the *Monte Carlo RL* and the *one-step bootstrapping* and allows shifting smoothly between them. On the other hand, the use of tabular RL methods for problems with large number of states is not always feasible because the problem converge only after having enough iterations over all the states. Also, gained knowledge can't be extrapolated to account for new states in case the environment of the problem is changed. An alternative to overcome these limitations is the use of *approximate solution methods* that approximate the value functions in several ways. One of these ways is to use artificial neural networks to approximate the Q-Values in Q-learning and this method is called *Deep Q-learning method* (Mnih et al., 2013).

5. Conclusion

The paper presents the method of combining Q-learning with Petri net model that forms an expert DSS that can be used to model and optimize SCS. The system was evaluated by considering a case study of SCS composed of 5 components that can be in one of 4 health states. The method showed the abil-

ity to optimize the maintenance schedule in a way to avoid catastrophic failures, reduce the maintenance costs, and reduce the system losses because of system's down time. The intelligent PN model created for the case study can be used to other SCS since the number of conditions for each component can be increased or decreased easily by changing the initial marking of place p_{1-n} of component n in Figure 2 that represent the health of component. The number of components of the system can be also changed by duplicating or removing the copies of the shaded region in Figure 2.

Acknowledgment

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.

References

David, R., & Alla, H. (2010). *Discrete, continuous, and hybrid Petri nets* (2nd ed.). Springer-Verlag Berlin Heidelberg.

De Dianous, V., & Fiévez, C. (2006). Aramis project: A more explicit demonstration of risk control through the use of bow–tie diagrams and the evaluation of safety barrier performance. *Journal of Hazardous Materials*, *130*(3), 220–233.

Drakaki, M., & Tzionas, P. (2017). Manufacturing scheduling using colored Petri nets and reinforcement learning. *Applied Sciences*, *7*(2), 136.

Dugan, J. B., Bavuso, S. J., & Boyd, M. A. (1992). Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on reliability*, *41*(3), 363–377.

Dunjó, J., Fthenakis, V., Vílchez, J. A., & Arnaldos, J. (2010). Hazard and operability (hazop) analysis. a literature review. *Journal of hazardous materials*, *173*(1-3), 19–32.

Kabir, S., & Papadopoulos, Y. (2019). Applications of bayesian networks and petri nets in safety, reliability, and risk assessments: A review. *Safety science*, *115*, 154–175.

Koch, M., Rust, C., & Kleinjohann, B. (2003). 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, pp. 217–222).

Leveson, N., Dulac, N., Marais, K., & Carroll, J. (2009). Moving beyond normal accidents and high reliability organizations: A systems approach to safety in complex systems. *Organization Studies*, *30*(2-3), 227-249. Retrieved from https://doi.org/10.1177/0170840608101478 doi: 10.1177/0170840608101478

Leveson, N. G., & Stolzy, J. L. (1987). Safety analysis using petri nets. *IEEE Transactions on software engineering*(3), 386–397.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), 541–580.

Palshikar, G. K. (2002). Temporal fault trees. *Information and Software Technology*, *44*(3), 137-150. doi: https://doi.org/10.1016/S0950-5849(01)00223-3

Stamatelatos, M., Vesely, W., Dugan, J., Fragola, J., Minarick, J., & Railsback, J. (2002). Fault tree handbook with aerospace applications.

Standard, M. (1980). *Procedures for performing a failure mode, effects and criticality analysis* (Tech. Rep.). MIL-STD-1629A.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3), 279–292.