Annals of Operations Research
https://doi.org/10.1007/s10479-022-04612-8

ORIGINAL RESEARCH

A deep reinforcement learning assisted simulated annealing algorithm for a maintenance planning problem

Fuat Kosanoglu[1] · Mahir Atmis[1] · Hasan Hüseyin Turan[2]

Accepted: 14 February 2022

Abstract
Maintenance planning aims to improve the reliability of assets, prevent the occurrence of asset failures, and reduce maintenance costs associated with downtime of assets and maintenance resources (such as spare parts and workforce). Thus, effective maintenance planning is instrumental in ensuring high asset availability with the minimum cost. Nevertheless, to find such optimal planning is a nontrivial task due to the (i) complex and usually nonlinear inter-relationship between different planning decisions (e.g., inventory level and workforce capacity), and (ii) stochastic nature of the system (e.g., random failures of parts installed in assets). To alleviate these challenges, we study a joint maintenance planning problem by considering several decisions simultaneously, including workforce planning, workforce training, and spare parts inventory management. We develop a hybrid solution algorithm ($\mathcal{DRLSA}$) that is a combination of Double Deep Q-Network based Deep Reinforcement Learning (DRL) and Simulated Annealing (SA) algorithms. In each episode of the proposed algorithm, the best solution found by DRL is delivered to SA to be used as an initial solution, and the best solution of SA is delivered to DRL to be used as the initial state. Different from the traditional SA algorithms where neighborhood structures are selected only randomly, the DRL part of $\mathcal{DRLSA}$ learns to choose the best neighborhood structure to use based on experience gained from previous episodes. We compare the performance of the proposed solution algorithm with several well-known meta-heuristic algorithms, including Simulated Annealing, Genetic Algorithm (GA), and Variable Neighborhood Search (VNS). Further, we also develop a Machine Learning (ML) algorithm (i.e., K-Median) as another benchmark in which different properties of spare parts (e.g., failure rates, holding costs, and repair rates) are used as clustering features for the ML algorithm. Our study reveals that the $\mathcal{DRLSA}$ finds the optimal solutions for relatively small-size instances, and it has the potential to outperform traditional meta-heuristic and ML algorithms.

Keywords Maintenance planning · Workforce planning and training · Inventory management · Double deep Q-network · Deep reinforcement learning · Simulated annealing

✉ Hasan Hüseyin Turan
h.turan@adfa.edu.au

[1] Department of Industrial Engineering, Yalova University, Yalova, Turkey

[2] Capability Systems Centre, University of New South Wales, Canberra, Australia

Published online: 15 March 2022

1 Introduction

Downtime is described as the accumulated amount of time when an asset is out of action or unavailable for use. It is required to keep the asset downtime under 10% to achieve world-class standards (Ong et al. 2020). Unplanned downtime due to asset failures can be very expensive. It is estimated that the average cost of unplanned downtime is $260,000 per hour across all industries (Arsenault 2016). Further, the downtime cost in terms of revenue lost per hour may vary depending on the industry. For example, while it is $2.1 million for pharmaceuticals, it reaches up to $4.6 million for the telecommunications industry (Hicks 2019).

Developing proper maintenance strategies is essential to reduce both the unplanned downtime and the operational costs of the assets. Maintenance strategies are broadly classified into two categories, namely, corrective maintenance and preventive maintenance (Rahmati et al. 2018). The corrective maintenance, which is also called breakdown maintenance, is performed after the failure and tries to bring the failed asset up to its operational status. Corrective maintenance is a reactive strategy. Thus, it can lead to high asset downtime and maintenance costs, if implemented improperly (Salari and Makis 2020). In this paper, we try to find a cost-effective implementation of the corrective maintenance by simultaneously optimizing different (but interconnected) planning decisions.

Among the aforementioned planning decisions, spare part inventory decisions have paramount importance since the unavailability of spare parts is the major reason for all asset downtime (Turan et al. 2020a; Kosanoglu et al. 2018). Holding a sufficient number of spare part stocks in inventory may reduce the asset downtime. However, many spare parts are expensive and it is economically more feasible to repair the failed parts/components of assets rather than replacing them with spares (Samouei et al. 2015; Levner et al. 2011). A failed part that can be repaired to an as-good-as-new condition is called a “repairable" part. In this paper, we optimize the number of repairable spare part stocks to keep in the inventory.

When a repairable part installed inside an asset fails, the defective part is substituted with a new one from the spare parts inventory (if there is any available) and the failed part is shipped to the maintenance/repair facility. In the maintenance facility, the failed part is repaired by the skilled maintenance workforce, afterwards, it is placed in spare stock inventory to be eventually used in case of another asset failure (Samouei et al. 2015; Turan et al. 2020b). It is clear that the maintenance workforce capacity (i.e., the number of technicians and their skills) in the repair facility directly affects the number of spare parts to keep in the inventory. In other words, a high number of skilled workforce would lead to faster repairs for failed parts and reduce throughput times in the maintenance facility, which would result in achieving the same availability level for assets with a less number of spare stocks. Thus, we include the maintenance workforce capacity optimization in our problem as the second planning decision.

It is generally assumed that all workforce in the maintenance facility has the same skill set and can repair all types of failed parts (i.e., full cross-training) in the system (Sleptchenko et al. 2019). Nevertheless, limited workforce flexibility with appropriate cross-training can offer most of the benefits of full cross-training (Jordan and Graves 1995). Moreover, if the repair tasks are complex and varied, utilizing full cross-trained maintenance workers may not be economical due to the limited learning capacity of workers and associated high training costs (Wang and Tang 2020). As our third maintenance planning decision, we aim at optimizing the skill assignment of the maintenance workforce. That is, we try to determine the skill set -in which each worker can only repair a subset of all part types- to minimize downtime with the minimum cost.

We contribute to the literature by addressing all the above-listed planning decisions in a single joint optimization model. Solving the modeled joint problem and finding the (near) optimal maintenance plans are nontrivial tasks due to the challenges associated with (i) the size of the decision space to search for (i.e., the number of feasible plans including, amount of spare stocks to keep on inventory for each part type, the number of maintenance workers to utilize in the maintenance facility and the number of possible combinations of assignments of skills to workers), and (ii) the stochastic nature of the system (e.g., random failures of parts installed in assets and service rate of maintenance).

To handle the challenge (i), we enhance a well-known meta-heuristic algorithm to search solution space more effectively. In this direction, we couple a Double Deep Q-Network based Deep Reinforcement Learning (DRL) and a Simulated Annealing (SA) algorithm. In this coupling, different from the traditional SA algorithms where neighborhood structures are selected only randomly, DRL learns to choose the best neighborhood structure based on experience gained from previous episodes and delivers the selected neighborhood structure to SA to use in future iterations. To overcome the challenge (ii), we model the maintenance facility as a queuing network. However, when the model's size gets larger (i.e., the number of spare part types and the number of workers), finding closed-form solutions for the modeled queuing network gets computationally demanding. Thus, we alleviate this difficulty by dividing the maintenance facility into smaller and independent repair cells where each repair cell is responsible for repairing a subset of all part types with its own cross-trained workforce.

The remainder of this paper is organized as follows. In Sect. 2, we review the relevant literature on deep reinforcement learning in maintenance problems. We also briefly discuss how deep reinforcement learning has been integrated with different optimization algorithms to solve hard optimization problems. In Sect. 3, the details of the problem are provided and the mathematical model is introduced. Next, we present the details of the solution algorithm in Sect. 4. We list and briefly discuss the benchmark algorithms and methods in Sect. 5. In Sect. 6, we outline the computational experiment design and present both computational and managerial results. Lastly, we discuss conclusions and future research directions in Sect. 7.

2 Literature Review

Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) have gained popularity in solving hard combinatorial optimization problems. In this section, we review the RL-DRL in combinatorial optimization and present a summarized literature review of the works in maintenance that employ RL-DRL as a solution approach in Table 1.

RL has shown promising results to solve complex combinatorial optimization problems. Integration of RL into optimization has been through two different patterns (Mazyavkina et al. 2020). In the first stream, the decision-maker can exploit the RL methods to directly find the part of the solution or the complete solution without any off-the-shelf solver (end to end learning) (Bengio et al. 2020). In this setting, RL has been employed to tackle many combinatorial optimization problems, such as Travelling Salesman Problem (TSP) (Bello et al. 2017; Kool et al. 2019; Emami and Ranka 2018; Ma et al. 2019; Deudon et al. 2018), Vehicle Routing Problem (VRP) (Nazari et al. 2018; Kool et al. 2019), and Bin Packing Problems (BPP) (Hu et al. 2017; Duan et al. 2019). Further examples of RL applications to optimization include traffic flow optimization (Walraven et al. 2016), pricing strategy optimization (Krasheninnikova and García 2019), bioprocess optimization (Petsagkourakis et al. 2020), and orienteering problem (Gama and Fernandes 2020).

Table 1 A summarized review of the literature on reinforcement-deep reinforcement learning in maintenance

Studies	RL-DRL algorithm	Application	Maintenance policy	Decision
Huang et al. (2020)	DDQN	Manufacturing	Preventive	Maintenance time and location
Andriotis and Papakonstantinou (2019)	PG,A3C	Civil engineering	Preventive	Replacement-repair
Zhang et al. (2019)	DQN, Model-Based	Manufacturing	Preventive	RUL estimation
Hoong Ong et al. (2020)	DDQN	Aero-engine	Preventive	Replacement-repair
Skordilis and Moghaddass (2020)	DQN	Aero-engine	Preventive	Maintenance time and RUL estimation
Rocchetta et al. (2019)	DQN	Energy	Preventive & Corrective	Operational and maintenance actions
Wei et al. (2020)	DQN	Civil engineering	Preventive	Replacement-repair
Yao et al. (2020)	DQN	Civil engineering	Preventive	Maintenance features
Li et al. (2019)	DQN	Aero-engine	Preventive	Replacement-repair
Allen et al. (2018)	BRL	Cybersecurity	Preventive	Actions against the risk
Zhang and Si (2020)	DQN	Generic	Condition-based	Replacement
This study	DDQN	Maintenance	Corrective	

PG: Policy Gradient, DQN: Deep Q Neural Network, DDQN: Double Deep Q Neural Network, A3C: Asynchronous Advantage Actor-Critic, BRL: Bayesian Reinforcement Learning

Another form is utilizing RL methods to improve the solution abilities of already existing solvers. In this approach, the optimization algorithm can call RL one time to initialize some parameter values or can call it repeatedly as the optimization algorithm iterates (Bengio et al. 2020). In this framework, RL methods are used to leverage the power of solvers or problem-specific solution heuristics by initializing values of some hyper-parameters. For example, RL can be utilized to select the branching variable in MIP solvers (Etheve et al. 2020; Hottung et al. 2020; Tang et al. 2020). Some recent studies of Ma et al. (2019); Deudon et al. (2018); Chen and Tian (2019) show that optimization heuristics powered with RL methods outperform previous methods.

Deep reinforcement learning (DRL) is one of the most intriguing areas of machine learning that combines reinforcement learning and deep learning. In particular, Deep Neural Network (DNN) allows reinforcement learning to be applied to complicated problems due to its ability to learn different levels of perception from data (François-Lavet et al. 2018). The benefits brought by DRL to other fields such as robotics and computer games are also employed to solve complex decision-making problems. Some recent works use DRL to solve some of the most prominent complex optimization problems such as resource management problems (Mao et al. 2016), job scheduling (Chen et al. 2017; Liu et al. 2020; Liang et al. 2020), VRP (Nazari et al. 2018; Lin et al. 2020; Zhao et al. 2020; Yu et al. 2019), and production scheduling problems (Waschneck et al. 2018b, a; Hubbs et al. 2020).

The maintenance planning (e.g., frequencies of corrective and preventive maintenance and condition-based maintenance policies) of spare parts inventory management models is well-studied in the literature with the various optimization models. Most of those studies employ off-the-shelf optimization software, heuristics, and well-known meta-heuristics (i.e., SA, Genetic Algorithm, Variable Neighborhood Search, etc.). Besides these methods, DRL is a promising new method in maintenance planning. Some recent studies employ RL to solve maintenance planning problems in various sectors. Huang et al. (2020) propose a preventive maintenance (PM) model in the Markov Decision Process (MDP) framework and solve the resulting problem employing the DRL algorithm. Andriotis and Papakonstantinou (2019) study a PM and inspection model for large multi-component engineering systems. They model a sequential decision-making problem with the MDP concept and propose a DRL algorithm that is capable of solving problems with massive state space.

Zhang et al. (2019) study the prediction of the equipment's remaining useful life, which is an essential indicator in maintenance planning. They model this problem as Health Indicator Learning (HIL) which learns a health curve that shows the equipment health conditions over time. Instead of conventional methods (inspection by hand or physical modeling) in addressing HIL, they propose a data-driven model that employs the DRL method. Hoong Ong et al. (2020) utilize sensor data to evaluate the equipment health condition and obtain optimal maintenance policy by DRL approach. Similarly, Skordilis and Moghaddass (2020) employ sensor data for a real-time decision-making framework for system maintenance based on DRL. Some other recent works use DRL to solve PM planning problems for different domains such as energy (Rocchetta et al. 2019), infrastructure (Wei et al. 2020; Yao et al. 2020), aero-engine (Li et al. 2019), and cybersecurity (Allen et al. 2018).

As IoT-based monitoring routine increases, intensive data-driven condition-based maintenance (CBM) planning gains increasing attention in recent years. Some recent studies show that using the benefits of RL techniques (such as learning from historical and online data) in CBM planning may accommodate promising results. Zhang and Si (2020) propose a CBM planning model for multi-component systems which can utilize equipment conditions at each inspection directly in decision-making without maintenance thresholds. Instead of computationally inefficient and intractable conventional threshold methods, they employ a

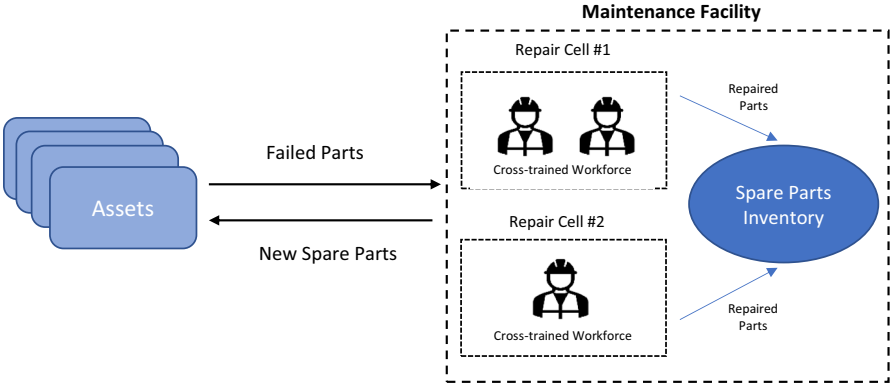


Fig. 1 The modeled maintenance planning problem

DRL-based method. Mahmoodzadeh et al. (2020) study an RL-based CBM planning model for corrosion-related maintenance management of dry gas pipelines.

The literature review in RL-DRL in maintenance optimization reveals the vacancy of research on the DRL-based solution approaches to corrective maintenance planning. This study contributes to maintenance planning optimization by developing a hybrid solution algorithm that combines Deep Reinforcement Learning (DRL) and Simulated Annealing (SA) algorithm.

3 Problem description and formulation

The studied maintenance planning problem consists of (i) a set of assets that includes components/parts subject to fail, (ii) a maintenance facility divided into repair cells, (iii) repair cells with a different number of cross-trained maintenance workforce that has different skill sets, and (iv) a spare parts stock point where multiple types of repairable spare parts; i.e., stock keeping units (SKUs), are kept as inventory. Fig. 1 provides high-level visualization of the studied problem.

Tables 2 and 3 list the problem parameters and decision variables used to formulate the mathematical model, respectively. We introduce the remaining parameters when they are needed.

We assume that assets contain N distinct types of parts (SKUs), and each SKU is subject to random failure throughout its lifetime with a constant rate λ_n $(n = 1, \ldots, N)$. We assume

Table 2 The list of problem parameters

N:	The number of distinct type of parts (SKUs) installed in assets
λ_n:	The failure rate of part type n $(n = 1, \ldots, N)$ that is in-use
μ_n:	The repair rate of part type n $(n = 1, \ldots, N)$ when it is failed
h_n:	The spare inventory holding cost for part type n $(n = 1, \ldots, N)$ per unit time per part
b:	The downtime cost per unit time when a failed part is backordered due to spare shortage in the inventory
α:	The base-level salary paid to a maintenance worker per unit time

Table 3 The list of decision variables

Decision variables:	
K:	The total number of repair cells in the maintenance facility
z_k:	The number of the maintenance workers in repair cell k $(k = 1, \ldots, K)$
I_n:	The amount of spare inventory (base stock level) kept on stock for part type n $(n = 1, \ldots, N)$
$x_{n,k}$:	A binary decision variable indicating that whether the part type n $(n = 1, \ldots, N)$ can be repaired in repair cell k $(k = 1, \ldots, K)$
Closed-form representations of decision variables	
$\mathbf{I}$:	A row matrix with $1 \times N$ dimensions donating the closed-form notation for the spare inventory levels kept on stock for each part type, where $\mathbf{I} = \left[I_1, \ldots, I_N\right]$
$\mathbf{X}$:	The closed-form matrix representation with $N \times K$ dimensions for the cross-training policy; i.e., the assignment of skills to repair cells. The element (n, k) in $\mathbf{X}$ corresponds to the binary decision variable $x_{n,k}$
$\mathbf{Z}$:	A row matrix with $1 \times K$ dimensions donating the closed-form notation for the number of the maintenance workers in each repair cell k $(k = 1, \ldots, K)$; i.e., the assignment of the workforce to cells, where $\mathbf{Z} = [z_1, \ldots, z_K]$

that the failure rate of the part is independent of its age (i.e., how long it has been in use). Further, we model the failure behavior of SKUs with exponential probability distributions with parameter λ_n $(n = 1, \ldots, N)$, which is a well-known assumption in repairable spare part inventory models (Sherbrooke 1968; Muckstadt 2005). We also assume the failure of an SKU is independent of other SKUs installed on the same asset.

When an in-use part fails, two things happen simultaneously: (i) an order is immediately placed for the same type of new (i.e., ready-for-use part) part to be supplied from spare stocks at the maintenance facility, and (ii) the failed part is sent to the maintenance facility as shown in Fig. 1.

The adopted spare inventory replenishment method mentioned in (i) corresponds to $(I_n - 1, I_n)$ policy. In this policy, the base stock inventory amount is equal to I_n for SKU type n $(n = 1, \ldots, N)$ and after each part failure, an order for a replacement is initiated for the same type of SKU. This is a well-known assumption used in repairable spare parts inventory systems (Sherbrooke 1986; Muckstadt 1973).

The failed part is shipped to the maintenance facility, and assigned to a repair cell k $(k = 1, \ldots, K)$. We assume that each part type n $(n = 1, \ldots, N)$ can be repaired at "only one" repair cell, and each repair cell k should be able to repair at least one type of SKU. Further, we assume that each repair cell k contains cross-trained workers z_k (where $z_k \geq 1$) that can repair any type of SKU assigned to their repair cell.

The failed parts that can be repaired in cell k form a single queue to be repaired by one of the maintenance workers in the cell (since all workforce in cell k can repair all types of SKUs assigned to that cell). We use the first come first served (FCFS) queuing discipline (i.e., there is no priority between failed parts), and the first available worker in the cell repairs the failed part. The repair time of failed part type n $(n = 1, \ldots, N)$ follows an exponential distribution with parameter μ_n $(n = 1, \ldots, N)$. The exponential distribution is often applied to maintenance tasks where the repair completion times are independent of previous maintenance operations, and durations of repair operations have high variability Turan et al. (2020a). In the literature, it is also shown via simulation studies that the maintenance systems are often insensitive to the repair time distributions (Sleptchenko and van der Heijden 2016; Sleptchenko et al. 2018) We also assume that the repair time is independent of the maintenance worker who is performing the repair given that the worker has the skill to repair the part n.

The objective of the model is to minimize the total cost $\mathcal{TC}$ by finding the (near) optimal settings of cross-training policy $\mathbf{X}$, the number of maintenance workers to allocate each repair cell $\mathbf{Z}$, and the number of spare stocks to keep in inventory for each part type $\mathbf{I}$. The objective of the studied maintenance planning problem is given in Eq. (1).

$$\mathcal{TC}[\mathbf{X}, \mathbf{Z}, \mathbf{I}] = \min_{\mathbf{X},\, \mathbf{Z},\, \mathbf{I}} \sum_{n=1}^{N} h_n I_n + \sum_{k=1}^{K} \alpha z_k + \sum_{k=1}^{K} z_k \left(\sum_{n=1}^{N} \beta_n x_{n,k} \right) + b \sum_{n=1}^{N} \mathbb{EBO}_n [\mathbf{X}, \mathbf{Z}, I_n] \tag{1}$$

The first summation term in Eq. (1) corresponds to the total inventory holding cost. This cost is calculated based on the initial stock levels for each part I_n, which is equal to the inventory position because of the implemented inventory replenishment method $(I_n - 1, I_n)$ (Turan et al. 2018; Sleptchenko et al. 2019). The second cost term reflects the cost paid to the workforce as the base-level salary. In addition to the base-level salary, a cross-trained workforce (that has the ability to repair one or more types of failed parts) can get the compensation payment depending on the number of skills the worker has. The compensation payment also includes the one-time training cost for workers. The third summation term in the objective function captures these costs.

When an order to replace the failed part can not be immediately met from the spare stocks at the maintenance facility, a backorder/backlog for the failed part occurs, and the asset goes down until a new part is supplied (either from the maintenance facility after the repair is completed or external part supplier). To reflect this, we calculate the "expected" number of backorders for SKU type n denoted by $\mathbb{EBO}_n[\cdot]$ as the function of decision variables $\mathbf{X}$, $\mathbf{Z}$, and $\mathbf{I}$. The total number of backorders (the last summation term in objective function) represents the number of assets that are down, and we multiply this value by the downtime cost per unit time per part on backorder b to obtain the total backorder cost (Turan et al. 2020c).

The decision variables have to satisfy some constraints to be a feasible solution to the modeled problem. First, the number of repair cells K in the maintenance facility cannot be less than one or more than N (in addition to being a positive integer $K \in \mathbb{Z}^+$). When the number of repair cells is equal to one, it is said that the maintenance facility is fully flexible/fully cross-trained, and in this type of facility each worker can fix all types of failures. In contrast, when the number of repair cells is equal to the number of distinct part types N, it is called the facility is fully dedicated. That is, each worker can only repair one type of part. Second, a feasible cross-training policy $\mathbf{X}$ has to be a member of the set defined in Eq. (2). More specifically, each SKU type has to be repaired at exactly one repair cell and each repair cell should be able to repair at least one SKU type.

$$\mathbf{X} \in \left\{ \begin{array}{l} \sum_{k=1}^{K} x_{n,k} = 1,\ n = 1, \ldots, N \\ \sum_{n=1}^{N} x_{n,k} \geq 1,\ k = 1, \ldots, K \\ x_{n,k} \in \{0, 1\},\ n = 1, \ldots, N\ k = 1, \ldots, K \\ 1 \leq K \leq N \ \text{ and } K \in \mathbb{Z}^+ \end{array} \right\} \tag{2}$$

Each repair cell k has to contain at least one worker ($z_k \geq 1,\ k = 1, \ldots, K$) and $z_k \in \mathbb{Z}^+$. Further, it should be ensured that the utilization of each repair cell k ($k = 1, \ldots, K$) has to be less than one in order to prevent the number of failed parts waiting to be served in each cell to go infinity as time progress. To achieve this, a sufficient number of maintenance workers has to be allocated to repair cell k in such a way that Eq. (3) holds. Lastly, the amount of spare inventory I_n kept on stock for SKU type n has to be a non-negative integer ($I_n \in \{0\} \cup \mathbb{Z}^+,\ n = 1, \ldots, N$).

$$z_k \geq \left\{ \begin{array}{ll} \sum_{n=1}^{N} x_{n,k} \frac{\lambda_n}{\mu_n} + 1, & \text{if} \sum_{n=1}^{N} x_{n,k} \frac{\lambda_n}{\mu_n} \text{ is integer} \\ \left\lceil \sum_{n=1}^{N} x_{n,k} \frac{\lambda_n}{\mu_n} \right\rceil, & \text{otherwise} \end{array} \right. , \ k = 1, \ldots, K \tag{3}$$

The number of feasible cross-training policy $\mathbf{X}$ that satisfies the set constraints in Eq. (2) for a given number of SKUs N and the number of repair cells K is the equivalent of the number of ways to partition a set of N distinct objects into K non-empty subsets. The latter is a well-known combinatorial problem with a solution equal to the Stirling number of the second kind (or Stirling partition number), which is denoted as $S(N, K)$ and calculated as

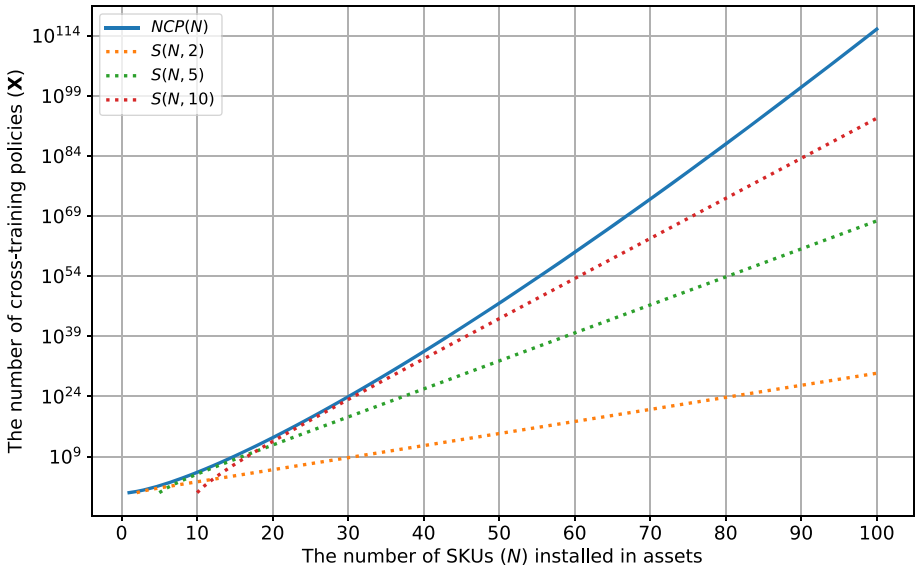


Fig. 2 The growth in the number of feasible cross-training policies (**X**) as a function of N and K

follows:

$$S(N, K) = \frac{1}{K!}\sum_{i=0}^{K}(-1)^i\binom{K}{i}(K-i)^N,\ K = 1, \ldots, N$$

Since K can take integer values between 1 and N, the exact total number of feasible cross-training policies $NCP(N)$ can be calculated as the function of N as follows:

$$NCP(N) = \sum_{K=1}^{N} S(N, K)$$

Figure 2 shows how $NCP(N)$ grows in the logarithmic scale, which proves the exponential increase in the size of decision space and the complexity of the problem. The size of the decision space gets even bigger when the decisions associated with the number of maintenance workers in each repair cell z_k and the spare inventory levels for each SKU type I_n are considered. Therefore, it is not possible or cumbersome to find the (near) optimal cross-training policy **X** with traditional optimization methods or a brute-force enumeration. To mitigate this, we enhance an SA algorithm with DRL to systematically check the promising policies and efficiently search the mentioned decision space.

4 Solution approach

In this section, we discuss the details of the solution approach. The approach basically contains three subroutines: a Simulated Annealing (SA) meta-heuristic, a Double Deep Q-Network-based Deep Reinforcement Learning (DRL) algorithm, and a simple greedy heuristic that uses a queuing approximation. These subroutines work collaboratively and iteratively to seek (near) optimal solutions for cross-training policy **X**, the number of maintenance workers to

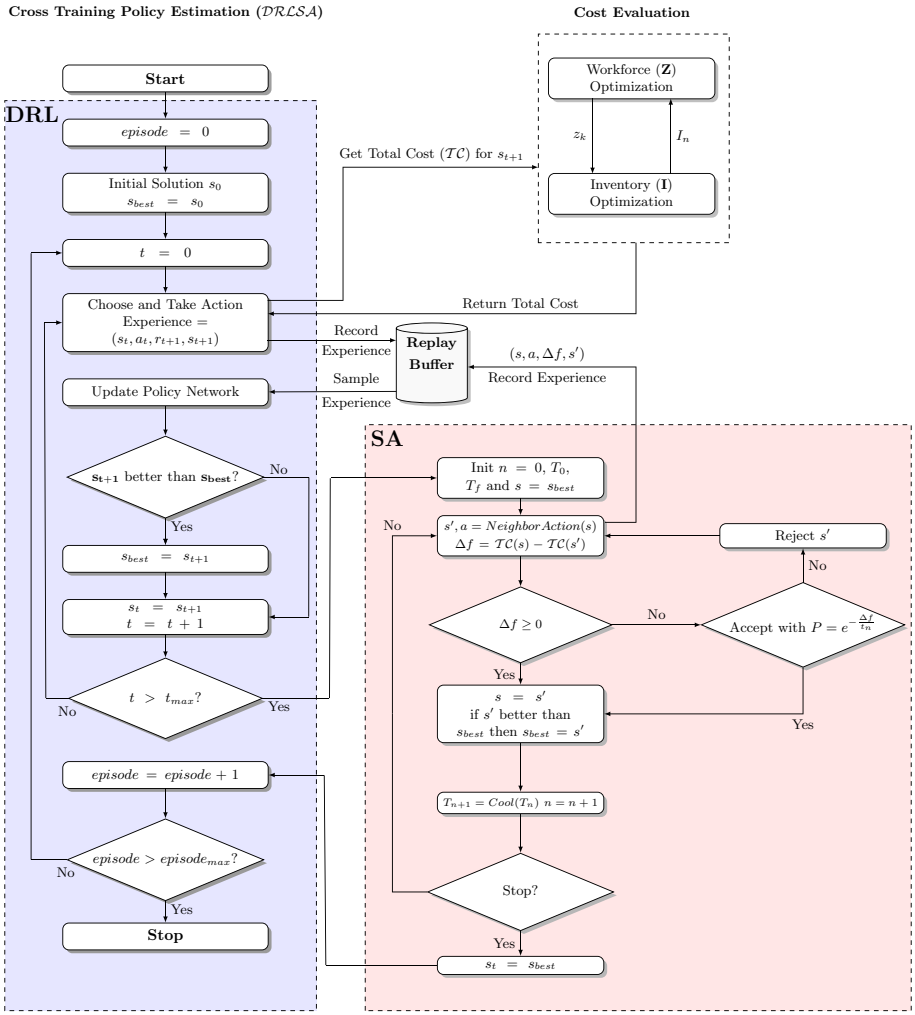


Fig. 3 Flow chart of the $\mathcal{DRLSA}$ Algorithm

allocate each repair cell **Z**, and the number of spare stocks to keep in inventory for each part type **I** as shown in Fig. 3.

We use a one-dimensional array encoding scheme to represent the solutions as shown in Fig. 4. The first part of the array shows the cross-training policy **X**. The length of this part (i.e., the number of cells) is equal to the number of part types (SKUs) N in the model. For the illustrative representation in Fig. 4, we have five SKUs in the model. For this part of the array, the number in each cell indicates the assignments of SKUs to repair cells. In the example, SKU 1, 2, and 5 are assigned to repair cell 1.

The second part of the array corresponds to the number of maintenance workers allocated to each repair cell **Z**. The length of this part (i.e., the number of cells) is dynamic and varies depending on the distinct numbers used in the first part of the array. In the example, the number of repair cells in maintenance facility K is two, and the first repair cell contains two

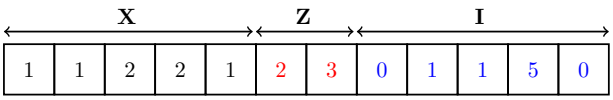


Fig. 4 An illustrative solution representation for a maintenance facility with 5 SKUs and 2 repair cells

and the second repair cell contains three workers. The last part of the array represents the number of spare stocks to keep in inventory for each part type **I**, and the length of this part is again equal to the number of part types (SKUs) N. The number in each cell denotes how many spares should be kept on stock for that SKU type. For example, for SKUs 1 and 5, it is not required to keep any spare stocks; i.e., $I_1 = I_5 = 0$. Solutions corresponding to Fig. 4 are shown in Eq. (4).

$$\mathbf{X} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \ \mathbf{Z} = [2, 3], \ \mathbf{I} = [0, 1, 1, 5, 0] \tag{4}$$

In the remainder of this section, we first discuss the key components of the reinforcement learning algorithm in Sect. 4.1. Section 4.2 explains why deep reinforcement learning is used. In Sect. 4.3, we give the details of the SA algorithm used, and in Sect. 4.4 we describe how we combined the ideas in DRL and SA. In Sect. 4.5, we discuss details of the greed heuristic that optimizes **Z** and **I**. Lastly, in Sect. 4.6, we list the value of all parameters that we used throughout the solution approach.

4.1 Reinforcement Learning (RL)

Reinforcement learning (RL) is a type of machine learning technique concerning how intelligent agents should take actions in an environment to maximize the cumulative reward (Hu et al. 2020). In a basic reinforcement algorithm, a decision-maker (also called an agent) is located in environment E. The agent sequentially interacts with the environment over time. At time step t, the agent gets the representation of the environment, which is called a state and denoted as $s_t \in S$, where S is the set of all possible states. Observing the state s_t, the agent chooses an action denoted as $a_t \in A$ to take under a policy π, where A denotes the set of all possible actions. Once the agent takes the action a_t, it gets a reward r_{t+1} and the environment transitions to a new state s_{t+1}. The goal of the agent under the policy π is to maximize its expected return G, while trying to reach its goal state as formulated in Eq. (5).

$$G_t^{\pi} = \sum_{k=0}^{\mathcal{T}} \gamma^k r_{t+k+1} \tag{5}$$

where $\mathcal{T}$ is the number of steps to take to reach the goal state from s_t and γ is the discount factor whose value is between [0, 1] indicating how much to discount future rewards. Lower values of γ significantly discount the future rewards while a value of 1 gives equal importance to all future rewards.

4.2 Deep Reinforcement Learning (DRL)

Deep reinforcement learning is the combination of reinforcement learning (RL) and deep learning. The term deep learning implies that an artificial neural network structure with more than 1 hidden layer is used while performing learning. Unfortunately, RL exhibits several limitations in practice when deployed in high-dimensional and complex stochastic domains, mainly manifesting algorithmic instabilities with solutions that significantly diverge from optimal regions, or exhibiting slow value updates at infrequently visited states (Andriotis and Papakonstantinou 2018). In such scenarios, deep learning methods are incorporated to solve the problem. The deep neural networks are strong function approximators. To benefit from their approximation capabilities in the reinforcement learning area, especially when the state-action space is very complex, they are used as policy function approximators as in Huang et al. (2020).

4.3 Simulated Annealing (SA)

Simulated annealing algorithm is initially introduced by Kirkpatrick (1984) which is one of the most popular and robust meta-heuristic algorithms that enable us to solve many hard combinatorial optimization problems (Suman and Kumar 2006) such as The Traveling Salesman Problem (TSP) (Kirkpatrick et al. 1983), and Quadratic Assignment Problem (QAP)(Connolly 1990). The SA uses a stochastic approach to avoid local optimum traps. In particular, SA searches for possible neighborhood solutions allowing a move to the neighboring solution even if the moved neighbor result is worse than the current one. In an analogy of SA with an optimization procedure, the physical material states correspond to problem solutions, the energy of a state to the cost of a solution, and the temperature to a control parameter (Du and Swamy 2016). The flowchart of the algorithm details can be found in the SA part of Fig. 3 which is highlighted in red. The algorithm starts with an initial feasible solution. At each iteration, SA generates a neighborhood solution and determines the objective function value based on this solution. If the resulting solution is better than the current solution then the new solution is accepted unconditionally. Otherwise, the neighborhood solution is accepted with the probability of $e^{(-\Delta E/T)}$ where T denotes the current temperature and ΔE denotes the difference between objective functions of current and the neighborhood solutions. Thus, the probability of accepting a worse solution is larger at high temperatures. The parameter of the temperature has crucial importance in the SA procedure since the acceptance probability is controlled with the temperature. This parameter gradually decreases at each iteration. Therefore, the probability of accepting uphill moves (accepting worse solution) is large at high T, and is low at low T.

4.4 $\mathcal{DRLSA}$

Our algorithm, denoted as $\mathcal{DRLSA}$, is a combination of Deep Reinforcement Learning (DRL) and Simulated Annealing (SA) algorithms.

4.4.1 State Space

In the $\mathcal{DRLSA}$, a state s_t represents the cross-training policy $\mathbf{X}$ mentioned in Sect. 3 and the finite state space S is any representation that $\mathbf{X}$ can get.

4.4.2 Action Space

The finite action space A is composed of the following three actions. The first action a^1, selects two random positions in the solution vector and swaps elements of these positions. The second action a^2, selects one element of the solution vector randomly and changes the value of this element. Similarly, the third action a^3, selects two elements of the solution vector randomly and changes the value of these elements.

4.4.3 Flow of the approach

In our implementation of the algorithm, both DRL and SA benefit from each other's feedback, as shown in Fig. 3. The DRL starts with a random initial state s_0 and runs for a number of time-steps, tries to learn which actions to take depending on the current state by changing its internal parameters, and passes the best state s_{best} it encounters during its training to SA. The SA then uses s_{best} as its initial solution, runs for several iterations and passes back its best solution s_{best} to DRL, and thus one episode becomes completed. The combined algorithm runs for a number of episodes, and the final s_{best} value, and its corresponding cost are reported as the final solution. The pseudo-code of the algorithm can be found in Algorithm 1 which in itself calls Algorithm 2.

Algorithm 1 Deep Reinforcement Learning Backed Simulated Annealing Algorithm ($\mathcal{DRLSA}$)

function $\mathcal{DRLSA}(s_0,\ obj,\ batch\_size,\ \gamma)$
  Initialize DQN, DQN_{target} networks
  $RB \leftarrow \{\}$ ▷ *Replay Buffer*
  Initialize $nTimeSteps, nEps, K$ ▷ Update target after K episodes
  $\mathcal{TC}_{best} \leftarrow obj(s_0)$
  $s_{best} \leftarrow s_0$
  for $i \leftarrow 1$ to $nEps$ **do**
    for $t \leftarrow 0$ to $\tau - 1$ **do**
      $a_t \leftarrow selectAction(s, DQN)$ ▷ Select using Epsilon(ε)-greedy policy
      $r_{t+1}, s_{t+1} \leftarrow takeAction(a_t)$
      $RB.push(s_t, a_t, r_{t+1}, s_{t+1})$
      $\mathcal{TC}_{new} \leftarrow obj(s_{t+1})$
      if $\mathcal{TC}_{new} <= \mathcal{TC}_{best}$ **then**
        $s_{best} \leftarrow s_{t+1}$
        $\mathcal{TC}_{best} \leftarrow \mathcal{TC}_{new}$
      end if
      if $len(RB) >= batch\_size$ **then**
        $s_t, a_t, r_{t+1}, s_{t+1} = RB.sample()$
        $Q_{current} \leftarrow getCurrentQ(s_t, a_t, DQN)$ ▷ Q-value of the network output for action a_t
        $Q_{next} \leftarrow getNextQ(s_{t+1}, DQN_{target})$ ▷ Maximum Q-value of the network output
        $Q^* \leftarrow Q_{next} * \gamma + r_{t+1}$
        $loss \leftarrow MSE\_Loss(Q_{current},\ Q^*)$
        Optimize DQN
      end if
      $s_t \leftarrow s_{t+1}$
    end for
    if $i \ \% \ K == 0$ **then**
      Set weight values of DQN_{target} to DQN's weight values
    end if
    $s_t, RB \leftarrow SA(s_{best}, obj, RB)$
  end for
  return s_{best}
end function

Algorithm 2 Simulated Annealing Algorithm (SA)

```
function SA(obj, RB)
    Initialize T_f, T_0, nIters
    T ← T_f
    a ← −log(T/T_0)
    TC_best ← obj(s)
    s_best ← s
    for t ← 1 to nIters do
        s′, a ← neighbor_and_action(s)        ▷ Find a neighbor and the action yielded there
        TC′ ← obj(s′)
        ΔE ← obj(s) − TC′
        RB.push(s, a, ΔE, s′)
        if ΔE <= 0 then
            s ← s′
            if TC′ <= TC_best then
                TC_best ← TC′
                s_best ← s
            end if
        else
            r ← rand(0, 1)
            if r < e^{−Δ/T} then
                s ← s′
            end if
        end if
    end for
    return s_best, RB
end function
```

4.4.4 Double Deep Q-Network

The type of the RL algorithm we used is Q-Learning (Watkins and Dayan 1992). Q-learning aims at learning the optimal action-value functions (also known as the Q-value functions or Q-functions) to derive the optimal maintenance management policy (Mahmoodzadeh et al. 2020). A Q-value, $Q(s_t, a_t)$ denotes how good it is to take the action a_t from state s_t. Traditional Q-learning uses a tabular structure(Q-table) to learn how to map action-state pairs to Q-values. Because of the complexity of the action-state space in our problem, using a Q-table-based solution wouldn't be feasible. Therefore, we adopted the Double Deep Q-Network described in Huang et al. (2020) which is derived from the basic deep Q-Network by Mnih et al. (2015) to approximate the Q-function. As depicted in Fig. 5, the first deep Q-network, denoted as DQN and is used as a policy network, takes a state s_t as input and produces Q-values for each possible action.

The optimal Q-function Q^* is formulated as in Eq. (6) which uses the Bellman optimality equation as an iterative update process to estimate the optimal Q-value for a given state-action pair (s_t, a_t).

$$Q^*(s_t, a_t) = \mathbb{E}\Big[r_{t+1} + \gamma \, max_{a_{t+1} \in A} \, Q^*(s_{t+1}, a_{t+1}) \big| \, s_t, a_t\Big] \tag{6}$$

The equation states that for a given state-action pair (s_t, a_t) at time t , the expected return of starting from state s_t, selecting action a_t and following the optimal policy thereafter will be the expected reward r_{t+1}, we get from taking action a_t in state s_t plus the maximum expected discounted return that can be achieved from any possible next state-action pairs (s_{t+1}, a_{t+1}).

The reward value r_{t+1} for $\mathcal{DRLSA}$ is defined as the difference between the total cost of the current state $obj(s_t)$, and the total cost of the next state $obj(s_{t+1})$. If the following state yields a lower cost value than the current state, the reward will be positive.

While selecting an action ($selectAction()$ in Algorithm 1) we use Epsilon(ε)-greedy strategy. The exploration rate ε is initially set to 1, which means the agent will explore the

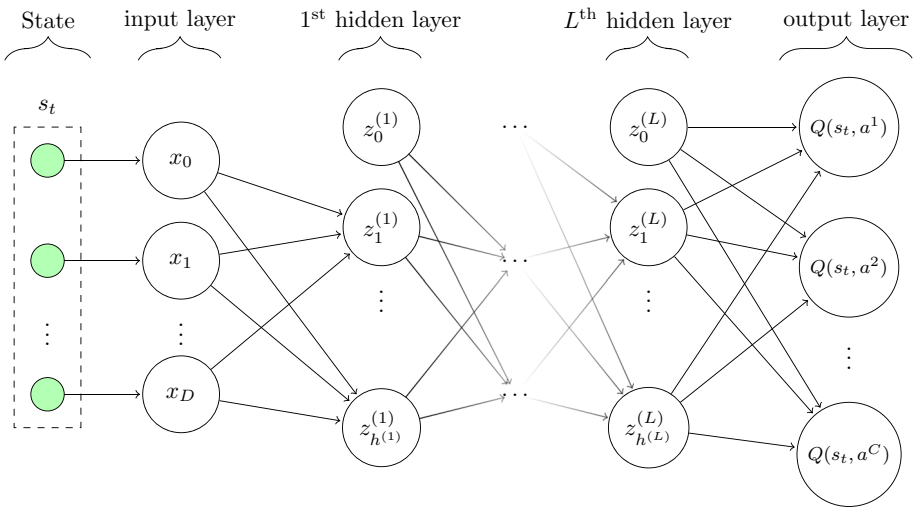


Fig. 5 Deep Q-network with L hidden layers D input units and C output units, where C denotes the number of possible actions. The l^{th} hidden layer contains $h^{(l)}$ hidden units. State s_t is fed as input to the network and Q-values for each action a^i, $Q(s_t, a^i)$, are predicted as output

environment rather than exploit it. As the agent learns more about the environment, ε gets decayed exponentially by some factor and the agent explores the environment with probability ε, and exploits it with probability $1 - \varepsilon$. Exploring the environment corresponds to choosing a random action while exploiting it means selecting the best action with maximum Q-Value, obtained by using DQN. We use a second Q-network called the target network, denoted as DQN_{target}, to calculate $Q^*(s_{t+1}, a_{t+1})$ part of Eq. (6). This type of architecture is called Double Deep Q-Network ($DDQN$) and leads to less overestimation of the Q-learning values, as well as improved stability, and hence improved performance (Hu et al. 2020).

The structure of DQN_{target} is the same as DQN, and they share the same parameters at first. The only difference is that during training, the parameters of DQN_{target} are not updated for a predefined number of episodes, η, while the parameters of the DQN are being updated at each iteration. After η episodes, the parameters of DQN_{target} are set to have the parameter values of the DQN. Equation (7) reformulates the optimal Q-Value in terms of networks.

$$Q^*(s_t, a_t) = r_{t+1} + \gamma \, max(DQN_{target}(s_{t+1}, a_{t+1})) \tag{7}$$

Since the Q-values produced by the network are real-valued numbers, the Mean Squared Error(MSE) loss function is used to train the network. It finds the error between the approximated optimal Q-value, $Q^*(s_t, a_t)$, and the Q-value of the current state-action pair, $Q(s_t, a_t)$ as formulated in Eq. (8).

$$L(s_t, a_t) = Q^*(s_t, a_t) - DQN(s_t, a_t) \tag{8}$$

To break the sequential behavior of the learning algorithm, we incorporated Experience Replay technique which was proved to improve the performance of the Q-networks. After each action is taken, the state, action, reward, and the next state value quadruple $(s_t, a_t, r_{t+1}, s_{t+1})$, called experience and denoted as ξ is stored in a queue named $Replay\ Buffer(RB)$. During each episode, both DRL and SA populate the RB with their experiences. At each iteration of the network training, if there are enough samples in the

buffer, a number of experiences are sampled from RB and are fed as input to the network to optimize the parameters of DQN.

The two important feedbacks that DRL gets from SA are the next state values and the experiences. Every time SA makes an iteration, it stores its experience ξ in DRL's RB so that DRL can use those experiences as a guide to update its parameters and learn how to predict which actions to take in upcoming episodes.

4.5 Optimizing workforce and spare part inventories for repair cells

We use a simple greedy heuristic to find the optimal values of workers to allocate each repair cell z_k ($k = 1, \ldots, K$) and base stock inventory levels I_n for each SKU type n ($n = 1, \ldots, N$) for a given cross-training policy $\mathbf{X}$ produced by $\mathcal{DRLSA}$ as shown in Fig. 3.

The greedy heuristic consists of two collaboratively acting subroutines. At the initial step, the workforce optimizer subroutine allocates the minimum number of workers required to each repair cell k by using:

$$z_k = \begin{cases} \sum_{n=1}^{N} x_{n,k} \frac{\lambda_n}{\mu_n} + 1, & \text{if } \sum_{n=1}^{N} x_{n,k} \frac{\lambda_n}{\mu_n} \text{ is integer} \\ \left\lceil \sum_{n=1}^{N} x_{n,k} \frac{\lambda_n}{\mu_n} \right\rceil, & \text{otherwise} \end{cases}, \quad k = 1, \ldots, K \tag{9}$$

which ensures the feasibility of Eq. (3).

Since repair cells are independent of each other based on the discussion provided by Turan et al. (2020b), Sleptchenko et al. (2019), we treat each repair cell k as a Markovian (due to exponential probability distributions for failure and service times) multi-class multi-server queuing systems $M/M/z_k$ where servers correspond to maintenance workers in repair cell k and classes correspond the set of SKU types that are assigned to cell k. The analysis of resulting queuing systems provides the expected number of backorders $\mathbb{EBO}_n[\cdot]$ for each SKU type n ($n = 1, \ldots, N$) which can be used to optimize I_n (see Turan et al. (2020c) for details). However, to evaluate the expected number of backorders $\mathbb{EBO}_n[\cdot]$ for larger repair cells (with the high number of workers z_k and part types N) is not computationally feasible, thus we use the aggregation-based approximation proposed by Van Harten and Sleptchenko (2003).

In each iteration of the greedy heuristic, the number of workers z_k in repair cell k increased by one to capture the trade-off between adding an additional worker to a cell and decreasing inventories of spares that are allocated to that cell. Iterations continue until employing an additional worker is not economical; i.e., the increased workforce doesn't lead to any spare inventory reduction.

4.6 Algorithm parameters and their values

The DRL parameters number of episodes($nEps$), and the number of time steps(τ) in Table 4 are chosen by using a grid search over the values of 100, 200, 300, 500, 1000, 2500 for $nEps$ and 32, 64, 128, 256, 512, and 1024 for τ. The values that yield in reasonable running time and close to optimum solutions are reported. The γ is chosen high not to discount the future rewards severely. Table 5 lists the used parameters for the deep Q-network. To control the stochastic behavior of the network and make training (which uses matrix calculations

Table 4 DRL parameters

Parameter name	Value
No. of Episodes ($nEps$)	300
No. of Time steps (τ)	128
Discount Factor (γ)	0.999
Epsilon (ε)	[0.01, 1]

Table 5 DQN parameters

Parameter name	Value
Batch Size ($batch\_size$)	32
Learning Rate (lr)	0.001
Optimizer	Adam
Target Network Update (η)	10
No. of Input Units (D)	{10, 20}
No. of Hidden Layers (L)	2
No. of Output Units (C)	3

Table 6 SA parameters

Parameter name	Value
Initial Temperature (T_0)	100,000
Final Temperature (T_f)	5
No. of Iterations ($nIters$)	1000

heavily), we used $batch\_size$ value of 32. The choice of the η value is also critical to ensure the target network catches up with the policy network. The common approach is to choose a value between 0.1 and 0.00001 (Wu et al. 2019). We adopted the recommended setting by Kandel and Castelli (2020). The number of input units D for DQN depends on the number of SKUs (N) used. Since the number of input units is at most 20 (see Sect. 6.1), only 2 hidden layers were enough to produce accurate results. The number of hidden units $h^{(l)}$ at l^{th} layer depends on D and is chosen using a grid search over possible values. The number of output units C for the network is equal to the number of available actions. In Table 6, we provide the important parameter values used for the SA algorithm. To update the parameters of the networks, we used Adam optimizer. We refer the reader to Kingma and Ba (2017) for details of the Adam optimizer.

5 Benchmark methods

We use two different sets of benchmark algorithms to compare the results of $\mathcal{DRLSA}$. The first set of benchmarks includes three well-known meta-heuristic algorithms, namely a Simulated Annealing (SA), a Genetic Algorithm (GA), and a Variable Neighborhood Search (VNS). The SA algorithm by Kosanoglu et al. (2018) is used as the benchmark, which is an improved version of the basic SA algorithm that we use in this study. Further, the details of benchmark GA and VNS algorithms together with their parameter settings can be found in Turan et al. (2020b).

Algorithm 3 A Generic K-Median Clustering

```
function Cluster(Costs, K, N)
    Randomly select K SKU's.
    Make them centroids c_k
    while true do
        for i ← 1 to K do
            Clusters[i] ← {}
        end for
        for i ← 1 to N do
            Distances ← {}
            for j ← 1 to K do
                Distances.append(Costs(i, c_j))
            end for
            index ← indexOfMin(Distances)
            Clusters[index].append(i)
        end for
        Calculate the medians of each cluster
        Make those medians new centroids
        If centroids didn't change break
    end while
    return Clusters
end function
```

Table 7 ML-based clustering algorithms variants

Algorithm 4 K-Median1

```
function KM1(μ, λ, K, h, N)
    μ ← μ/max(μ)                                  ▷ Scale to [0,1] range
    h ← h/max(h)
    λ ← λ/max(λ)
    for i ← 0 to N − 1 do
        for j ← i + 1 to N − 1 do
            C(i, j) ← 1/3 ∗ [(h_i − h_j)^2 + (λ_i − λ_j)^2 + (μ_i − μ_j)^2]
        end for
    end for
    return Cluster(C, K, N)
end function
```

Algorithm 5 K-Median2

```
function KM2(μ, K, N)
    for i ← 0 to N − 1 do
        for j ← i + 1 to N − 1 do
            C(i, j) ← (μ_i − μ_j)^2
        end for
    end for
    return Cluster(C, K, N)
end function
```

Algorithm 6 K-Median3

```
function KM3(λ, K, N)
    for i ← 0 to N − 1 do
        for j ← i + 1 to N − 1 do
            C(i, j) ← (λ_i − λ_j)^2
        end for
    end for
    return Cluster(C, K, N)
end function
```

Algorithm 7 K-Median4

```
function KM4(K, h, N)
    for i ← 0 to N − 1 do
        for j ← i + 1 to N − 1 do
            C(i, j) ← (h_i − h_j)^2
        end for
    end for
    return Cluster(C, K, N)
end function
```

In the second set of benchmarks, we consider a machine learning-based (ML-based) clustering algorithm with four variants (K-Median1, K-Median2, K-Median3, K-Median4). We use a K-Median Clustering as provided in Algorithm 3. Variant algorithms differentiate from each other in terms of clustering features they employ. We use different properties of parts (e.g., failure rates λ_n, holding costs h_n, and repair rates μ_n) as clustering features. Table 7 presents pseudo-codes for each variant are provided in Algorithms 4, 5, 6, and 7.

6 Computational Study

To evaluate the performance of the proposed $\mathcal{DRLSA}$ approach, we conduct a detailed numerical analysis. Particularly, in Sect. 6.1, we define the experiment testbed employed

Table 8 Problem parameter variants for testbed

Factors	Levels
No. of SKUs (N)	10, 20
No. of initial workers (M)	5, 10
Utilization rate (ρ)	0.65, 0.80
Minimum holding cost (h_{min})	1, 100
Maximum holding cost (h_{max})	1000
Holding cost/workload relation	IND, HPB
Workforce cost (α)	$10h_{max}$, $100h_{max}$
Compensation payment or Cross-training cost (β_n)	0.01α 0.10α
Backorder cost (b)	$50\frac{\sum_{n=1}^{N}\lambda_n h_n}{\sum_{n=1}^{N}\lambda_n}$

in our analysis. Section 6.2 explores the solution quality of the approach by comparing it with total enumeration results. Then, Sect. 6.3 presents a detailed run time and convergence analysis. Next, in Sect. 6.4, we compare the total system cost achieved by the proposed $\mathcal{DRLSA}$ algorithm with the costs achieved by benchmark algorithms. Finally, in Sect. 6.5, we investigate capacity usage and cross-training policies.

6.1 Testbed

We utilize the same testbed of instances as in Turan et al. (2020c) to investigate the effectiveness of the proposed $\mathcal{DRLSA}$ method. In this testbed, a full factorial design of experiment (DoE) with seven factors and two levels per factor is used as shown in Table 8. We test the proposed algorithm and all benchmarks with a total of 128 instances. In this testbed, initial workers M denotes the minimum number of workers that have to be utilized in the maintenance facility. Further, we use two different patterns, namely completely random and independent (IND) and hyperbolically related (HPB) to generate the holding costs h_n. The HPB pattern reflects the cases in which expensive repairables are repaired less frequently. For the explicit definitions and derivation of test instances and factors, we refer the reader to the work of Turan et al. (2020c).

6.2 Optimality gaps and optimal solution behavior

In this section, we give an analysis for assessing the solution quality of the proposed $\mathcal{DRLSA}$ algorithm. We employ the testbed given in Sect. 6.1 to investigate the gap between $\mathcal{DRLSA}$ solutions and the optimal solutions. In order to create relatively small size test problems that can be solved by a brute force (total enumeration) approach, we randomly choose four, five, six, and seven SKUs from each case generated in the previous section. First, we investigate the gap between the minimum cost achieved by total enumeration (i.e. the optimal cost) and the $\mathcal{DRLSA}$ algorithm. The $\mathcal{DRLSA}$ algorithm achieves the optimal cost for all tested 512 (4×128) cases. We further investigate the optimal solution behavior. In particular, we inspect how the optimal number of repair cells (clusters) K differs as the number of SKUs N changes. We present the number of repair cells for each solved case in Fig. 6. We observe that

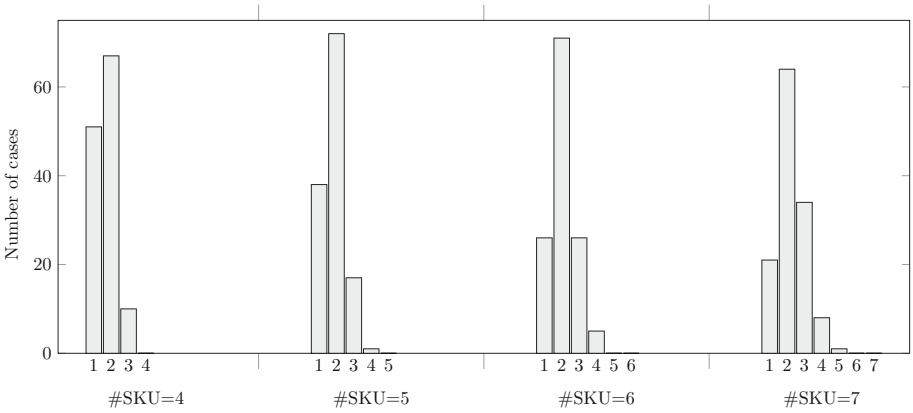


Fig. 6 The number of repair cells K for each optimally solved cases

the optimal number of repair cells never reaches the number of SKUs N (such that dedicated design). Our results also indicate that the maintenance facility is divided into mostly two repair cells regardless of the number of SKUs N.

6.3 Runtime and convergence analysis

In this subsection, we comment on the convergence and runtime of $\mathcal{DRLSA}$, and SA and RL stages. We implemented all algorithms discussed in this paper in Python programming language and ran them on a desktop computer with 4 core 3.60 GHz CPU and 8 GB RAM.

Table 9 shows the effect of the problem factors on runtime. Under all problem factors, we observe that most of the runtime (nearly 85%) is occupied by SA rather than RL stages. It should be also noted that on average runtime of $\mathcal{DRLSA}$ increases nearly twice when Cross-training cost (β_n) is reduced to 0.01 from 0.10. Further, another noticeable runtime fluctuation occurs when the minimum holding cost (h_{min}) is increased from 1 to 100.

For illustrative purposes, the total cost $\mathcal{TC}$ convergence is shown for a single case in Fig. 7. In this particular case, $\mathcal{DRLSA}$ starts with a sharp decrease in total cost at early episodes (until around episode 10) and continues to converge to the best solution until around Episode 260. Figure 7 also presents how cost convergence occurs in a single episode between RL and SA stages. For the case we presented, it is clear that the SA stage does not perform well after iteration 600, which may indicate that initial temperature T_o might be reduced without affecting solution quality.

6.4 Performance comparisons

In this section, the performance of the proposed algorithm is evaluated by comparing it with a set of benchmark algorithms described in Sect. 5. We present a detailed analysis of cost reductions achieved by the proposed $\mathcal{DRLSA}$ algorithm and factors affecting the performance.

We first assess how $\mathcal{DRLSA}$ performs in cost compared to benchmark algorithms. Figure 8 presents pair-wise performance comparisons of $\mathcal{DRLSA}$ to benchmark algorithms. In particular, we present the number of cases $\mathcal{DRLSA}$ outperforms, underperforms, and per-

Table 9 Runtime analysis for each stage of $\mathcal{DRLSA}$

Factors	Levels	RL Stage (CPU sec.)				SA Stage (CPU sec.)				$\mathcal{DRLSA}$ Episode (CPU sec.)			Total runtime	SA runtime share (%)	RL runtime share (%)
		max.	mean	min.	iteration	max.	mean	min.	iteration	max.	mean	min.			
No. of SKUs *(N)*	10	110.11	40.31	21.25	0.31	890.02	294.39	155.92	0.29	928.65	334.71	188.69	100414.03	87.41	12.58
	20	111.49	76.59	24.36	0.59	601.85	392.01	246.73	0.39	685.31	468.60	283.15	140581.85	83.51	16.48
No. of initial workers *(M)*	5	102.28	62.23	21.55	0.48	610.61	318.68	202.17	0.31	681.68	380.92	237.34	114277.79	84.01	15.98
	10	119.32	54.66	24.06	0.42	881.26	367.73	200.48	0.36	932.28	422.39	234.51	126718.09	86.91	13.08
Utilization rate (ρ)	0.65	81.20	47.39	22.01	0.37	633.19	302.26	190.56	0.30	682.99	349.65	221.57	104896.21	86.76	13.24
	0.80	140.40	69.51	23.59	0.54	858.68	384.15	212.08	0.38	930.97	453.66	250.28	136099.67	84.16	15.83
Minimum holding cost (h_{min})	1	83.23	51.28	20.19	0.40	550.34	259.04	169.88	0.26	608.20	310.32	202.55	93096.80	84.26	15.73
	100	156.75	70.40	27.15	0.55	1071.93	483.48	253.73	0.48	1138.28	553.88	291.55	166166.50	87.46	12.53
Holding Cost/Work Load	HPB	136.16	67.37	24.56	0.52	886.25	365.47	198.27	0.36	954.31	432.84	237.43	129852.27	84.00	15.99
	IND	85.44	49.53	21.04	0.38	605.62	320.94	204.38	0.32	659.65	370.47	234.42	111143.61	86.92	13.07
Workforce cost (α)	$10h_{max}$	140.02	61.77	27.28	0.48	985.17	379.40	211.88	0.37	1046.17	441.18	259.95	132354.92	85.53	14.46
	$100h_{max}$	81.58	55.12	18.33	0.43	506.70	307.01	190.76	0.30	567.78	362.13	211.90	108640.96	85.39	14.60
Cross-training cost (β_n)	0.01α	149.90	73.38	25.74	0.57	1112.02	458.42	239.43	0.45	1187.71	531.81	275.02	159544.04	86.19	13.81
	0.10α	71.70	43.51	19.87	0.34	379.84	227.99	163.21	0.22	426.24	271.50	196.83	81451.84	84.73	15.26

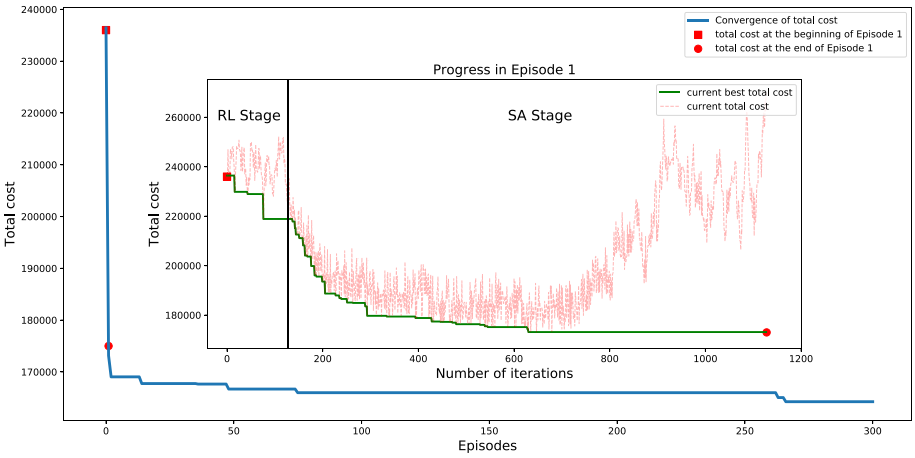


Fig. 7 The convergence behaviour of $\mathcal{DRLSA}$ for a single case

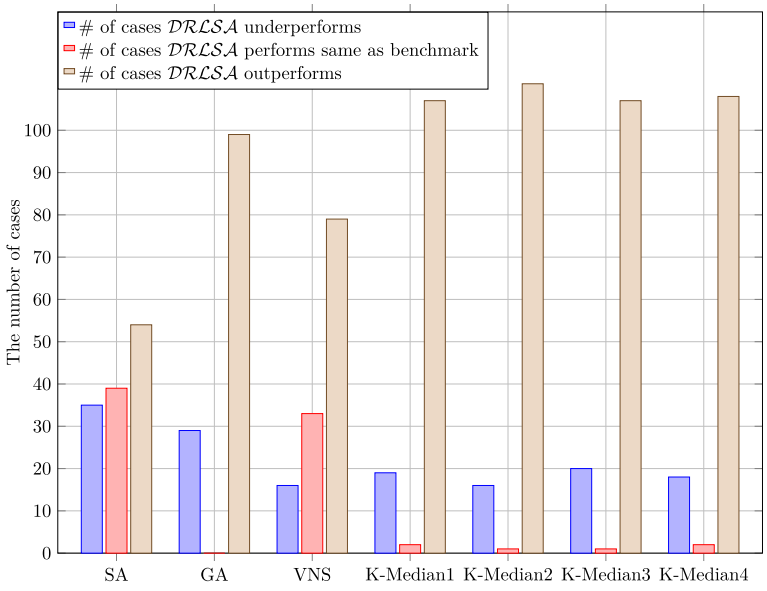


Fig. 8 Pair-wise performance comparisons of $\mathcal{DRLSA}$ to benchmark algorithms

forms the same as benchmark algorithms in achieving minimum cost. In most of the cases, the $\mathcal{DRLSA}$ algorithm is superior to benchmark algorithms. The second best performing algorithm is SA which outperforms the $\mathcal{DRLSA}$ in 35 out of 128 cases, while the $\mathcal{DRLSA}$ is superior in 54 out of 128 cases. Table 10 presents the number of cases lowest cost achieved by each algorithm under each of the problem factors. For each case, multiple algorithms may achieve the minimum cost. The $\mathcal{DRLSA}$ is the best performing algorithm in achieving the minimum cost 55 out of 128 cases, followed by SA with 39 out of 128 cases. GA algorithm achieves the lowest total cost in 29 of 128 cases, VNS algorithm achieves the lowest total

cost in 20 of 128 cases, and K-Median1, K-Median2, K-Median3, K-Median4 algorithms achieve the lowest total cost in 15, 11, 16, 13 of 128 cases respectively. We also observe that fully flexible design achieves the lowest total cost in 11 of 128 cases, and dedicated designs never achieve the minimum cost. We also observe that GA and VNS algorithms are extremely sensitive to the number of SKUs N while SA and $\mathcal{DRLSA}$ are relatively less sensitive. Another notable result is the increase in workforce cost/base-level salary (α) increases the performance of $\mathcal{DRLSA}$ and VNS, decreases GA, and doesn't affect SA. In general, SA is the least sensitive algorithm to the problem factors.

We next investigate how the total cost achieved by $\mathcal{DRLSA}$ compares to the benchmark algorithms. Figure 9 presents the total cost of $\mathcal{DRLSA}$ minus the total cost of the benchmark algorithm. The minimum costs achieved by SA and VNS are relatively close to $\mathcal{DRLSA}$.

In Figs. 10a and b, we present cost reductions achieved by each algorithm in comparison to dedicated and fully flexible designs, respectively. The cost reduction amounts are sensitive to problem factors in comparison to both dedicated and fully flexible design. In particular, all algorithms attain the highest cost reduction compared to the dedicated design when N is large (20), M is small (5), workforce cost is large ($100h_{max}$), and cross-training cost is small (0.01α). Presumably, when N is large, dedicated design results in lower utilization of workers. Similarly, when workforce cost is higher, the cost of dedicated design results in a larger cost due to the extensive number of workers. When M is small, the additional workforce demand increases the cost due to the requirements of the dedicated design. Finally, when cross-training cost is small instead of having many maintenance workers dedicated to a single SKU type, cross-training workforce are more cost-effective. On the other hand, the cost reduction is more sensitive to cross-training costs in comparison to fully flexible design due to an increase in training cost (compensation payment) of workers for each SKU. We can also remark that cost reduction of SA, VNS, GA, and $\mathcal{DRLSA}$ are substantially larger than ML-based algorithms.

We also evaluate the performance of $\mathcal{DRLSA}$ for larger problem instances with 50 and 100 SKUs. We compare the achieved objective function values of $\mathcal{DRLSA}$ and ML-based algorithms. We observe that $\mathcal{DRLSA}$ achieves the best cost reduction in comparison to fully flexible design followed by K-median3, K-median2, K-median1, and K-median4 in 50 SKUs case. Nevertheless, in 100 SKUs case, K-median3 achieves the best cost reduction followed by $\mathcal{DRLSA}$, K-median2, K-median1, and K-median4. The detailed results are presented in Table 11. The performance of the $\mathcal{DRLSA}$ algorithm may be improved for larger SKU numbers (larger search space) by increasing the iteration and episode numbers. Furthermore, the proposed $\mathcal{DRLSA}$ algorithm provides the solutions for the larger maintenance planning problems in an acceptable time.

6.5 Capacity usage and cross-training analysis

This section investigates how repair cell formation and cross-training depend on problem parameters and solution algorithms. In Table 12, we present the average number of workers utilized per case, the average number of repair cells in the optimized solution, and the average number of skills per worker. Interestingly, the number of repair cells formed with ML-based algorithms is extremely sensitive to cross-training cost factor levels, yet SA, VNS, GA, and $\mathcal{DRLSA}$ are relatively less sensitive to cross-training cost factor levels. Specifically, when the cross-training cost increased from 0.01α to 0.1α, the average number of repair cells is nearly doubled. We also observe that the numbers of repair cells formed with SA, VNS, GA, and $\mathcal{DRLSA}$ are more sensitive to the number of SKUs N (on average 75% increase in the

Table 10 The number of cases lowest cost achieved by each algorithm under each of the problem factors

Factors	Levels	SA	Flexible	Dedicated	VNS	GA	$\mathcal{DRLSA}$	K-Median1	K-Median2	K-Median3	K-Median4
No. of SKUs *(N)*	10	19	2	0	16	29	33	2	2	2	3
	20	20	9	0	4	0	22	13	9	14	10
No. of initial workers *(M)*	5	17	10	0	12	10	30	12	10	14	11
	10	222	1	0	8	19	25	3	1	2	2
Utilization rate (ρ)	0.65	19	4	0	8	16	27	6	4	8	4
	0.8	20	7	0	12	13	28	9	7	8	9
Minimum holding cost (h_{min})	1	19	6	0	7	18	25	8	6	7	7
	100	20	5	0	13	11	30	7	5	9	6
Holding Cost/Work Load Relation	HPB	20	5	0	10	14	26	7	5	7	7
	IND	18	6	0	10	15	29	8	6	9	6
Workforce cost (α)	$10h_{max}$	19	10	0	8	18	18	11	10	12	10
	$100h_{max}$	20	1	0	12	11	37	4	1	4	3
Cross-training cost (β_n)	0.01α	16	11	0	8	12	24	15	11	16	13
	0.10α	23	0	0	12	17	31	0	0	0	0

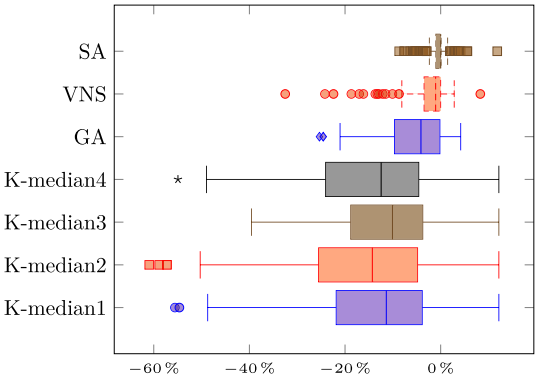
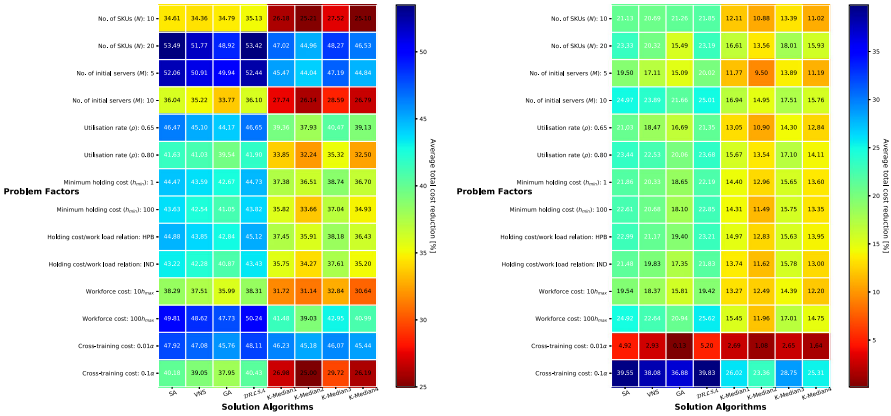


Fig. 9 Pair-wise total cost comparison of $\mathcal{DRLSA}$ and benchmark algorithms



(a) Dedicated design vs. all solution algorithms

(b) Fully flexible design vs. all solution algorithms

Fig. 10 Average cost reductions in comparison with dedicated and fully flexible designs

Table 11 The achieved percentage cost reductions in larger instances in comparison to fully flexible design

# of SKUs	$\mathcal{DRLSA}$	K-median1	K-median2	K-median3	K-median4
50	6.36	3.57	5.53	6.21	3.45
100	9.07	8.86	8.98	12.93	8.34

average number of repair cells formed when the number of SKUs is increased from 10 to 20). Surprisingly, the numbers of repair cells formed with K-Median1 and K-Median4 decrease as the number of SKUs increases, while a slight increase is observed for K-Median2 and K-Median3.

Another question we explore is how much cross-training would be enough under various problem factors. Figure 11 shows the average cross-training percentages achieved by each algorithm for each problem factor. Our results demonstrate that the average cross-training percentages of ML-based algorithms are larger compared to other algorithms. Although the differences are not large, the cross-training percentage order of other algorithms is $\mathcal{DRLSA}$, SA, VNS, and GA from the largest to the smallest, respectively, for almost all problem factors. The average cross-training percentage is highly sensitive to cross-training costs β_n

Table 12 Average # of repair cells, average # of worker per cell, and average # of skill per worker under each of the problem factors

Factors	Levels	SA			VNS			GA			$\mathcal{DRLSA}$		
		nC	nSPC	nSPS	nC	nSPC	nSPS	nC	nSPC	nSPS	nC	nSPC	nSPS
No. of SKUs *(N)*	10	3.64	7.16	3	3.78	7.25	2.92	3.56	7.19	3.21	3.33	7.02	3.45
	20	5.91	7.89	3.66	6.48	8.45	3.21	6.84	8.73	3.14	5.88	7.72	3.6
No. of initial workers *(M)*	5	4.42	5.56	3.52	4.81	5.89	3.2	4.73	5.94	3.52	4.19	5.34	3.82
	10	5.13	9.48	3.14	5.45	9.81	2.93	5.67	9.98	2.84	5.02	9.39	3.23
Utilization rate (ρ)	0.65	4.61	6.94	3.43	5.08	7.36	3.05	5.06	7.47	3.23	4.41	6.78	3.63
	0.8	4.94	8.11	3.23	5.19	8.34	3.08	5.34	8.45	3.12	4.8	7.95	3.42
Minimum holding cost (h_{min})	1	4.84	7.48	3.2	5.13	7.83	3.07	5.23	7.84	3.16	4.61	7.33	3.57
	100	4.7	7.56	3.46	5.14	7.88	3.06	5.17	8.08	3.2	4.59	7.41	3.48
Holding Cost/Work Load Relation	HPB	4.97	7.45	3.13	5.23	7.78	2.97	5.25	7.83	3.05	4.77	7.31	3.37
	IND	4.58	7.59	3.53	5.03	7.92	3.16	5.16	8.09	3.31	4.44	7.42	3.68
Workforce cost (α)	$10h_{max}$	4.72	8.17	3.57	5.13	8.47	3.21	5.19	8.59	3.32	4.7	8	3.61
	$100h_{max}$	4.83	6.88	3.09	5.14	7.23	2.92	5.22	7.33	3.04	4.5	6.73	3.44
Cross-training cost (β_n)	0.01α	4.08	7.39	3.97	4.39	7.61	3.64	4.53	7.81	3.78	3.91	7.28	4.25
	0.10α	5.47	7.66	2.69	5.88	8.09	2.49	5.88	8.11	2.57	5.3	7.45	2.81
Factors	Levels	K-Median1			K-Median2			K-Median3			K-Median4		
		nC	nSPC	nSPS	nC	nSPC	nSPS	nC	nSPC	nSPS	nC	nSPC	nSPS
No. of SKUs *(N)*	10	3.08	7.86	5.45	3.3	8	5.5	2.86	7.67	5.53	3.33	7.92	5.09
	20	3.63	8.27	10.79	3.86	8.73	12.17	3.89	8.2	9.23	3.16	8.13	11.28
No. of initial workers *(M)*	5	2.72	5.7	9.05	2.91	5.98	9.48	2.77	5.59	8.43	2.56	5.69	9.23
	10	3.98	10.42	7.19	4.25	10.75	8.2	3.98	10.28	6.33	3.92	10.36	7.15
Utilization rate (ρ)	0.65	3.25	7.41	8.44	3.58	7.81	8.8	3.16	7.27	7.84	3	7.34	8.37
	0.8	3.45	8.72	7.81	3.58	8.92	8.87	3.59	8.61	6.92	3.48	8.7	8
Minimum holding cost (h_{min})	1	3.58	8.02	7.94	3.42	8.13	8.8	3.41	7.81	7.55	3.22	7.97	8.4
	100	3.13	8.11	8.31	3.73	8.61	8.87	3.34	8.06	7.21	3.27	8.08	7.98
Holding Cost/Work Load Relation	HPB	3.3	7.94	7.86	3.48	8.17	8.97	3.38	7.83	7.36	3.3	8	7.76
	IND	3.41	8.19	8.38	3.67	8.56	8.71	3.38	8.05	7.39	3.19	8.05	8.62
Workforce cost (α)	$10h_{max}$	3.53	8.75	8.51	3.61	8.92	8.87	3.17	8.47	8.36	3.19	8.67	8.73
	$100h_{max}$	3.17	7.38	7.74	3.55	7.81	8.81	3.58	7.41	6.4	3.3	7.38	7.64
Cross-training cost (β_n)	0.01α	1.56	7.2	12.19	1.44	7.33	13.79	1.72	7.27	11.12	1.56	7.27	12.39
	0.10α	5.14	8.92	4.06	5.72	9.41	3.88	5.03	8.61	3.64	4.92	8.78	3.98

Note: nC: average # of repair cells, nSPC: average # of used workers, nSPS: average # of skill per worker.

for all algorithms. More specifically, the average cross-training percentages of ML-based algorithms are tripled as cross-training costs increased from 0.01α to 0.1α, whereas cross-training percentages of other algorithms are nearly doubled. The number of SKUs N is another essential problem factor that affects the average cross-training percentage. In particular, when the number of SKUs is increased from 10 to 20, the average cross-training percentage is diminished by half for $\mathcal{DRLSA}$ and other meta-heuristics. However, an increase is observed for K-Median1, K-Median2, and K-Median4, and a decrease for K-Median3 as the number of SKUs increases 10 to 20. This result arguably arises due to the clustering features of

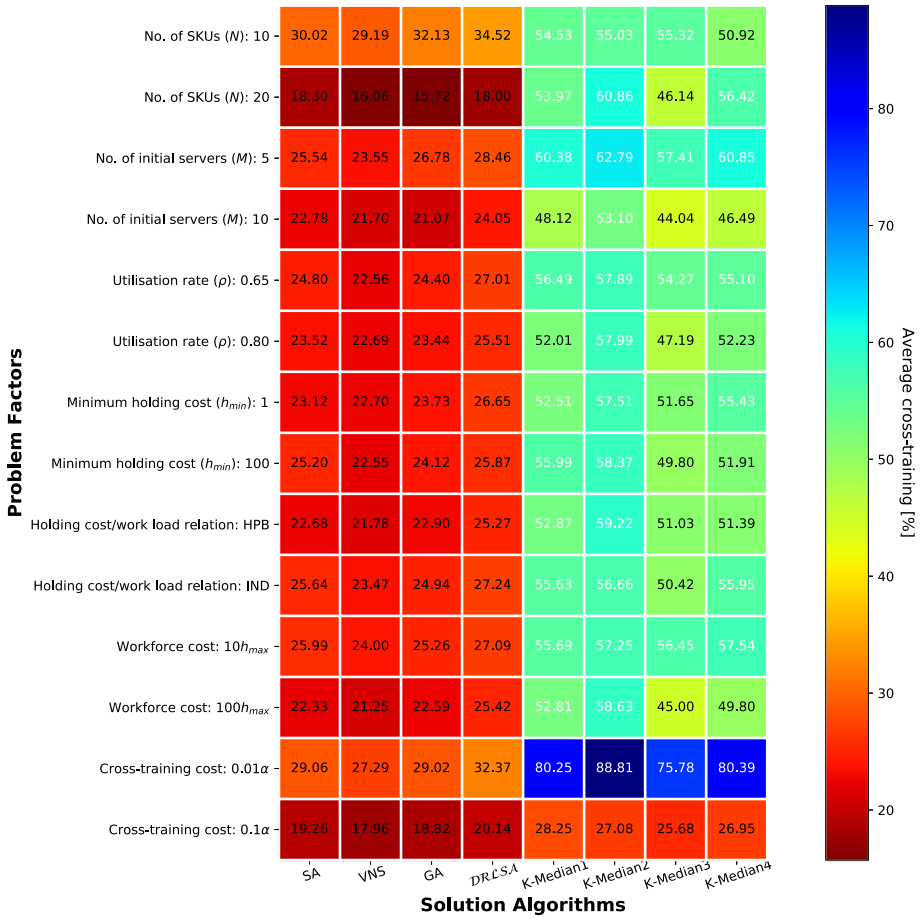


Fig. 11 The average cross-training percentages obtained by each algorithm for each problem factors

algorithms. K-Median3 uses the failure rate of the in-use part as a clustering feature (λ_n), while other algorithms use the repair rate of the failed part (μ_n), the inventory holding cost (h_n), or both.

7 Conclusions and Future Research

The design of effective maintenance planning is crucial to ensure high asset availability with the minimum cost. In this paper, we develop a heuristic algorithm inspired by DRL to solve a joint maintenance planning problem by taking into account several decisions simultaneously, including workforce planning, workforce training, and spare parts inventory management. This work is an initial attempt to improve solution algorithms using DRL for corrective maintenance problems.

The computational results show that the proposed solution algorithm is promising in solving the defined problem. Particularly, the proposed $\mathcal{DRLSA}$ algorithm obtains a bet-

ter objective function value (total cost) than well-known meta-heuristic algorithms (SA, GA, VNS) and machine learning-based clustering algorithms (K-Median1, K-Median2, K-Median3, and K-Median4). Our work reveals that there exists a great potential in the application of machine learning to optimization. In particular, machine learning-enforced optimization algorithms may provide relatively better solutions. We note that the performance of algorithms is sensitive to problem parameters. Moreover, parameters that affect the performance of algorithms may differ for each algorithm. For example, the increase in workforce cost/base-level salary (α) increases the performance of $\mathcal{DRLSA}$ and VNS, decreases GA, but does not affect SA.

The studied joint maintenance planning problem has a few major limitations. First, it is assumed that the failure rates of parts are constant throughout their lifetime. However, a more realistic modeling would consider the aging of parts in-use and adjust the failure rates as a function of time. Second, the model is analyzed under static policies (e.g. routing of failed parts in the maintenance facility) to ensure the computational tractability of the queuing approximations.

All of the above-listed computational challenges arising due to the analysis of queuing models could be alleviated by modeling repairable supply chains (including the repair shop) via a simulation model and coupling this simulation model with DRL. This is a niche area in the current literature that enables decision-maker to analyze and optimize dynamic problems under uncertainty. Further, another possible extension could be improving the $\mathcal{DRLSA}$ algorithm by exploiting the great potentials of the other reinforcement learning methods such as SARSA (State-Action-Reward-State-Action) to increase solution quality while reducing the runtime. Additionally, it might be worthwhile to integrate the DRL algorithm with other meta-heuristics such as GA and VNS.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions.

References

Allen, T. T., Roychowdhury, S., & Liu, E. (2018). Reward-based Monte Carlo-Bayesian reinforcement learning for cyber preventive maintenance. *Computers & Industrial Engineering, 126*, 578–594.

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

Andriotis, C. P., & Papakonstantinou, K. G. (2018). Managing engineering systems with large state and action spaces through deep reinforcement learning. *CoRR*, arXiv:1811.02052.

Arsenault, R. (2016). Stat of the week: The (rising!) cost of downtime. https://www.aberdeen.com/techpro-essentials/stat-of-the-week-the-rising-cost-of-downtime/. Accessed: 2021-03-07.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2017). Neural combinatorial optimization with reinforcement learning. arXiv:1611.09940.

Bengio, Y., Lodi, A., & Prouvost, A. (2020). Machine learning for combinatorial optimization: a methodological tour d'horizon. arXiv:1811.06128.

Chen, W., Xu, Y., & Wu, X. (2017). Deep reinforcement learning for multi-resource multi-machine job scheduling. arXiv preprint arXiv:1711.07440, .

Chen, X., & Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. arXiv:1810.00337.

Connolly, D. T. (1990). An improved annealing scheme for the QAP. *European Journal of Operational Research, 46*, 93–100.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., & Rousseau, L.-M. (2018). Learning heuristics for the TSP by policy gradient. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (pp. 170–181). Springer.

Du, K.-L., & Swamy, M. N. S. (2016). Simulated annealing. *Search and Optimization by Metaheuristics: Techniques and Algorithms Inspired by Nature* (pp. 29–36). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-41192-7_2

Duan, L., Hu, H., Qian, Y., Gong, Y., Zhang, X., Xu, Y., & Wei, J. (2019). A multi-task selected learning approach for solving 3D flexible bin packing problem. arXiv:1804.06896.

Emami, P., & Ranka, S. (2018). Learning permutations with sinkhorn policy gradient. arXiv:1805.07010.

Etheve, M., Alès, Z., Bissuel, C., Juan, O., & Kedad-Sidhoum, S. (2020). Reinforcement learning for variable selection in a branch and bound algorithm. *Lecture Notes in Computer Science*, (p. 176–185).

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. Foundations and Trends&#x00AE. *Machine Learning, 11*, 219–354.

Gama, R., & Fernandes, H. L. (2020). A reinforcement learning approach to the orienteering problem with time windows. arXiv:2011.03647.

Hicks, G. (2019). How much is equipment downtime costing your workplace? https://www.iofficecorp.com/blog/equipment-downtime. Accessed: 2021-03-07.

Hoong Ong, K. S., Niyato, D., & Yuen, C. (2020). Predictive maintenance for edge-based sensor networks: A deep reinforcement learning approach. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)* (pp. 1–6). https://doi.org/10.1109/WF-IoT48130.2020.9221098.

Hottung, A., Tanaka, S., & Tierney, K. (2020). Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers & Operations Research, 113*, 104781. https://doi.org/10.1016/j.cor.2019.104781 http://www.sciencedirect.com/science/article/pii/S0305054819302230.

Hu, H., Zhang, X., Yan, X., Wang, L., & Xu, Y. (2017). Solving a new 3D bin packing problem with deep reinforcement learning method. arXiv:1708.05930.

Hu, J., Niu, H., Carrasco, J., Lennox, B., & Arvin, F. (2020). Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. *IEEE Transactions on Vehicular Technology, 69*, 14413–14423. https://doi.org/10.1109/TVT.2020.3034800

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

Hubbs, C. D., Li, C., Sahinidis, N. V., Grossmann, I. E., & Wassick, J. M. (2020). A deep reinforcement learning approach for chemical production scheduling. *Computers & Chemical Engineering, 141*, 106982.

Jordan, W. C., & Graves, S. C. (1995). Principles on the benefits of manufacturing process flexibility. *Management Science, 41*, 577–594.

Kandel, I., & Castelli, M. (2020). The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express, 6*, 312–315. https://doi.org/10.1016/j.icte.2020.04.010 https://www.sciencedirect.com/science/article/pii/S2405959519303455.

Kingma, D.P., & Ba, J. (2017). Adam: A method for stochastic optimization. arXiv:1412.6980.

Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics, 34*, 975–986.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*, 671–680.

Kool, W., van Hoof, H., & Welling, M. (2019). Attention, learn to solve routing problems! arXiv:1803.08475.

Kosanoglu, F., Turan, H. H., & Atmis, M. (2018). A simulated annealing algorithm for integrated decisions on spare part inventories and cross-training policies in repairable inventory systems. In *Proceedings of International Conference on Computers and Industrial Engineering* (pp. 1–14).

Krasheninnikova, E., & García, J., Maestre, R., & Fernández, F. (2019). Reinforcement learning for pricing strategy optimization in the insurance industry. *Engineering Applications of Artificial Intelligence, 80*, 8–19. https://doi.org/10.1016/j.engappai.2019.01.010 http://www.sciencedirect.com/science/article/pii/S0952197619300107.

Levner, E., Perlman, Y., Cheng, T., & Levner, I. (2011). A network approach to modeling the multi-echelon spare-part inventory system with backorders and interval-valued demand. *International Journal of Production Economics, 132*, 43–51.

Li, Z., Zhong, S., & Lin, L. (2019). An aero-engine life-cycle maintenance policy optimization algorithm: Reinforcement learning approach. *Chinese Journal of Aeronautics, 32*, 2133–2150.

Liang, S., Yang, Z., Jin, F., & Chen, Y. (2020). Data centers job scheduling with deep reinforcement learning. In H. W. Lauw, R.C.-W. Wong, A. Ntoulas, E.-P. Lim, S.-K. Ng, & S. J. Pan (Eds.), *Advances in Knowledge Discovery and Data Mining* (pp. 906–917). Cham: Springer International Publishing.

Lin, B., Ghaddar, B., & Nathwani, J. (2020). Deep reinforcement learning for electric vehicle routing problem with time windows. arXiv:2010.02068.

Liu, C., Chang, C., & Tseng, C. (2020). Actor-critic deep reinforcement learning for solving job shop scheduling problems. *IEEE Access, 8*, 71752–71762. https://doi.org/10.1109/ACCESS.2020.2987820

Ma, Q., Ge, S., He, D., Thaker, D., & Drori, I. (2019). Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. arXiv:1911.04936.

Mahmoodzadeh, Z., Wu, K.-Y., Droguett, E. L., & Mosleh, A. (2020). Condition-based maintenance with reinforcement learning for dry gas pipeline subject to internal corrosion. *Sensors, 20*, 5708.

Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (pp. 50–56).

Mazyavkina, N., Sviridov, S., Ivanov, S., & Burnaev, E. (2020). Reinforcement learning for combinatorial optimization: A survey. arXiv:2003.03600.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*, 529–533. https://doi.org/10.1038/nature14236

Muckstadt, J. A. (1973). A model for a multi-item, multi-echelon, multi-indenture inventory system. *Management Science, 20*, 472–481.

Muckstadt, J. A. (2005). *Analysis and algorithms for service parts supply chains*. Germany: Springer Science & Business Media.

Nazari, M., Oroojlooy, A., Snyder, L. V., & Takáč, M. (2018). Reinforcement learning for solving the vehicle routing problem. arXiv:1802.04240.

Ong, K. S. H., Niyato, D., & Yuen, C. (2020). Predictive maintenance for edge-based sensor networks: A deep reinforcement learning approach. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)* (pp. 1–6). IEEE.

Petsagkourakis, P., Sandoval, I., Bradford, E., Zhang, D., & del Rio-Chanona, E. (2020). Reinforcement learning for batch bioprocess optimization. *Computers & Chemical Engineering, 133*, 106649. http://www.sciencedirect.com/science/article/pii/S0098135419304168.

Rahmati, S. H. A., Ahmadi, A., & Govindan, K. (2018). A novel integrated condition-based maintenance and stochastic flexible job shop scheduling problem: simulation-based optimization approach. *Annals of Operations Research, 269*, 583–621.

Rocchetta, R., Bellani, L., Compare, M., Zio, E., & Patelli, E. (2019). A reinforcement learning framework for optimal operation and maintenance of power grids. *Applied Energy, 241*, 291–301.

Salari, N., & Makis, V. (2020). Joint maintenance and just-in-time spare parts provisioning policy for a multi-unit production system. *Annals of Operations Research, 287*, 351–377.

Samouei, P., Kheirkhah, A. S., & Fattahi, P. (2015). A network approach modeling of multi-echelon spare-part inventory system with backorders and quantity discount. *Annals of Operations Research, 226*, 551–563.

Sherbrooke, C. C. (1968). Metric: A multi-echelon technique for recoverable item control. *Operations Research, 16*, 122–141.

Sherbrooke, C. C. (1986). VARI-METRIC: Improved approximations for multi-indenture, multi-echelon availability models. *Operations Research, 34*, 311–319.

Skordilis, E., & Moghaddass, R. (2020). A deep reinforcement learning approach for real-time sensor-driven decision making and predictive analytics. *Computers & Industrial Engineering, 147*, 106600.

Sleptchenko, A., Hanbali, A. A., & Zijm, H. (2018). Joint planning of service engineers and spare parts. *European Journal of Operational Research, 271*, 97–108.

Sleptchenko, A., & van der Heijden, M. (2016). Joint optimization of redundancy level and spare part inventories. *Reliability Engineering & System Safety, 153*, 64–74.

Sleptchenko, A., Turan, H. H., Pokharel, S., & ElMekkawy, T. Y. (2019). Cross-training policies for repair shops with spare part inventories. *International Journal of Production Economics, 209*, 334–345.

Suman, B., & Kumar, P. (2006). A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society, 57*, 1143–1160. https://doi.org/10.1057/palgrave.jors.2602068

Tang, Y., Agrawal, S., & Faenza, Y. (2020). Reinforcement learning for integer programming: Learning to cut. arXiv:1906.04859.

Turan, H. H., Atmis, M., Kosanoglu, F., Elsawah, S., & Ryan, M. J. (2020a). A risk-averse simulation-based approach for a joint optimization of workforce capacity, spare part stocks and scheduling priorities in maintenance planning. *Reliability Engineering & System Safety, 204*, 107199.

Turan, H. H., Kosanoglu, F., & Atmis, M. (2020b). A multi-skilled workforce optimisation in maintenance logistics networks by multi-thread simulated annealing algorithms. *International Journal of Production Research,* 1–23. https://doi.org/10.1080/00207543.2020.1735665

Turan, H. H., Sleptchenko, A., Pokharel, S., & ElMekkawy, T. Y. (2018). A clustering-based repair shop design for repairable spare part supply systems. *Computers & Industrial Engineering, 125*, 232–244.

Turan, H. H., Sleptchenko, A., Pokharel, S., & ElMekkawy, T. Y. (2020c). A sorting based efficient heuristic for pooled repair shop designs. *Computers & Operations Research, 117*, 104887.

Van Harten, A., & Sleptchenko, A. (2003). On Markovian multi-class, multi-server queueing. *Queueing systems, 43*, 307–328.

Walraven, E., Spaan, M. T., & Bakker, B. (2016). Traffic flow optimization: A reinforcement learning approach. *Engineering Applications of Artificial Intelligence*, 52, 203 – 212. http://www.sciencedirect.com/science/article/pii/S0952197616000038. https://doi.org/10.1016/j.engappai.2016.01.001.

Wang, Y., & Tang, J. (2020). Optimized skill configuration for the seru production system under an uncertain demand. *Annals of Operations Research*, (pp. 1–21).

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., & Kyek, A. (2018a). Deep reinforcement learning for semiconductor production scheduling. In *2018 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)* (pp. 301–306). https://doi.org/10.1109/ASMC.2018.8373191.

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., & Kyek, A. (2018b). Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP*, 72, 1264 – 1269. 51st CIRP Conference on Manufacturing Systems.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*, 279–292. https://doi.org/10.1007/BF00992698

Wei, S., Bao, Y., & Li, H. (2020). Optimal policy for structure maintenance: A deep reinforcement learning framework. *Structural Safety, 83*, 101906.

Wu, Y., Liu, L., Bae, J., Chow, K.-H., Iyengar, A., Pu, C., Wei, W., Yu, L., & Zhang, Q. (2019). Demystifying learning rate policies for high accuracy training of deep neural networks. arXiv:1908.06477.

Yao, L., Dong, Q., Jiang, J., & Ni, F. (2020). Deep reinforcement learning for long-term pavement maintenance planning. *Computer-Aided Civil and Infrastructure Engineering*, 35, 1230–1245. arXiv:https://onlinelibrary.wiley.com/doi/pdf/101111/mice.12558.

Yu, J. J. Q., Yu, W., & Gu, J. (2019). Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems, 20*, 3806–3817. https://doi.org/10.1109/TITS.2019.2909109

Zhang, C., Gupta, C., Farahat, A., Ristovski, K., & Ghosh, D. (2019). Equipment health indicator learning using deep reinforcement learning. In U. Brefeld, E. Curry, E. Daly, B. MacNamee, A. Marascu, F. Pinelli, M. Berlingerio, & N. Hurley (Eds.), *Machine Learning and Knowledge Discovery in Databases* (pp. 488–504). Cham: Springer International Publishing.

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

Zhao, J., Mao, M., Zhao, X., & Zou, J. (2020). A hybrid of deep reinforcement learning and local search for the vehicle routing problems. *IEEE Transactions on Intelligent Transportation Systems* (pp. 1–11). https://doi.org/10.1109/TITS.2020.3003163