Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 103 (2021) 1–7

www.elsevier.com/locate/procedia

9th CIRP Global Web Conference – Sustainable, resilient, and agile manufacturing and service operations: Lessons from COVID-19

A Deep Reinforcement Learning Based Scheduling Policy for Reconfigurable Manufacturing Systems

Jiecheng Tang, Konstantinos Salonitis *

[a]*Sustainable Manufacturing Systems Centre, Cranfield University, Bedford, MK43 0AL, UK*

* Corresponding author. Tel.: +44 (0) 1234 758347. *E-mail address:* k.salonitis@cranfield.ac.uk

Abstract

Reconfigurable manufacturing systems (RMS) is one of the trending paradigms toward a digitalised factory. With its rapid reconfiguring capability, finding a far-sighted scheduling policy is challenging. Reinforcement learning is well-equipped for finding highly efficient production plans that would bring near-optimal future rewards. For minimising reconfiguring actions, this paper uses a deep reinforcement learning agent to make autonomous decision with a built-in discrete event simulation model of a generic RMS. Aiming at the completion of the assigned order lists while minimising the reconfiguration actions, the agent outperforms the conventional first-in-first-out dispatching rule after self-learning.

Keywords: Reconfigurable manufacturing system; scheduling; reinforcement learning; dueling double deep q learning; discrete event simulation

1. Introduction

Aiming at rapidly addressing demand fluctuation, the reconfigurable manufacturing system (RMS) was first introduced in 1999 [1]. RMS paradigm targets on medium-volume-high-variant mixed orders. With its structural adjusting capability, RMS challenges the conventional Dedicated Manufacturing Line (DML) on product variation [2] and scalable setup [3]. RMS also challenges Flexible Manufacturing Systems (FMS) on productivity and cost [1]. To classify the challenge into details, six core characteristics has been identified [4] for RMS that include modularity, integrability, diagnosability, scalability, customisability, and convertibility. Napoleone [5] investigated the interrelationship between these characteristics. Perceived value of a product lead to the emerging of massive customisation, the top level characteristic of RMS. Tang et al. [6] pointed out that a dynamic scheduling policy which can continuously evolve in a full sensing digital twin environment could help improving customisability by enhancing scalability and convertibility. However, the uncertainty, caused by the changing demand, scale, and constrains of manufacturing practice, keeps challenging this NP-hard [7] job-shop scheduling problem. Mathematical programming and metaheuristic algorithms [8] are the main tools to solve similar scheduling problems for FMS. These methods outperform the fixed dispatching rules like first-in-first-out but underperform while tackling the scalability of RMS with limited computing resources [4]. RMS needs a “far-sighted” and “resilient” control strategy. These features make reinforcement learning (RL) outstanding and particularly feasible to cope similar continuous and dynamic flexible job-shop scheduling problems [9]. In the present paper, a discrete-event simulation (DES) model, which simulates a simplified RMS, provides a Markov decision process environment for training a job releasing schedule agent. By observing the current system state, this agent sends

10.1016/j.procir.2021.09.089

an action command to the DES model to obtain state transition and a reward. Using continuous state updating and rewarding, this scheduling agent trains one value approximator which then provides an optimal scheduling policy. The rest of this paper is organised as follows: section two formulates the problem by reviewing the relative research on RMS scheduling, RL in scheduling optimisation, optimal paths of RL training, and RL-enabled DES. Section three describes the proposed RMS-DES workflow, explaining how data flow between the RL agent and DES model environment, and how the agent trains its neural networks and uses it as a value approximator. Section four presents a numerical case study operated in different setups for optimising general production time of an RMS by minimising the reconfiguring actions. Section five finally concludes that this framework could easily train an intelligent agent outperform conventional FIFO dispatch rule.

2. Background and Literature Review

This paper mainly focuses on optimising RMS scheduling using Deep Q Learning (DQL) by reducing reconfiguring actions and while minimising the makespan. Q-learning [10] based optimisation is used when the problems can be formulated as Markov Decision Processes (MDP). In short, an MDP means system state changes can only be affected by the last chosen action. Q-learning is a value-based policy training strategy. In an MDP environment, an agent has limited action options from an action-space for a certain state. Every action perform on that MDP environment leads to a state-transition. A value function estimates the expected reward feedback from the MDP environment. This expected reward from a particular action is the quality, Q, of this action. Serial actions, based on the highest Q-value, form an optimal policy. Then, labelling these actions with time forms an optimal schedule. For a simple environment with small number of states, all rewards due to the state-action pairs can be calculated and recorded in a tabulated database. This recording process is called Q-learning. State-action pairs quantity grows exponentially in a complex system. In such case, a neural network [11] is introduced as a powerful tool with limited input dimensions (state), limited output dimensions (actions), and estimating value for each output (expected reward). This deep neural network approximating Q value is named Deep Q-network (DQN). The training process to make DQN estimating the expected reward closer to the accurate reward from MDP is called Deep-Q-learning (DQL) [12] which extends from Q-learning. Discrete-event simulation (DES) behaves similar to an MDP on state transition which makes it an appropriate environment training a smart agent. Aydin and Öztemel [13] trained an agent, through a revised version of Q learning, to solve a dynamic job-shop scheduling problem. The MDP process is formulated as a DES where the agent needs to choose one of fixed dispatching rules at certain scheduling time. These literatures suggested DES is an appropriate method to deal with "what-if" problem. Solving job-shop scheduling problem using DES is not a new approach. For decades, scholars have focused on several significant open-shop scheduling problems [14] including m-machine, batch processing and setup time, resource-constrained project scheduling problem, release time and on-line scheduling, and so on. Fixed dispatching rules are the most common control policy of traditional production lines aiming on making scheduling fast. RMS paradigm and the evolving Industry 4.0 technologies bring extra dimensions for observing the system while raising additional considerations in previous scheduling problems. These added dimensions help operators understand the system better while, at the same time, make the mature methods underperform. In scheduling, conventional methods are incapable for complex system such as RMS [15–18] and advanced adaptive scheduling is needed to achieve a near optimal control solution. Improving system robustness is another scheduling research topic focused on optimal scheduling using reinforcement learning. Palombarini et al. [19] used an intelligent agent with DQL to reschedule a socio-technical manufacturing system in real time with new inserted tasks disturbance. Tackling on a resource-constrained scheduling problem, He et al. [20] proposed a two-stage DQN framework and suggested that RL has great potential in solving complex task assigning problems. Huang et al. [21] obtained a preventive maintenance policy on a DML by a DRL approach. Though most scheduling research focuses on DML, a shared patten could inspire and be implemented on RMS to achieve high-level decision-making autonomy [22] and tackle the computational difficulties of such complex system. Different from the sequential layout in DMLs, another inspiring research field is parallel machine scheduling problem [9]. Fixed dispatching policies, such as first in first out (FIFO) and earliest due date (EDD), are major comparisons for popular meta-heuristic multi-objectives optimisation methods like NSGA-II [23]. Using an intelligent agent to find optimal operating policies attracted researchers attention soon after Sutton and Barto [24] provided an excellent reinforcement learning background knowledge in 1998. Creighton et al. [25] used state-action map (Q-learning) technique to optimising operation policy on a multi-part DML. Gabel et al [26] formulated a job-shop scheduling problem into a decentralised Markov decision process and successfully gained optimal scheduling control by applying gradient-descent methods. Waschneck et al [27] applied DQN in multiple machines for a global production scheduling problem by training these agents sequentially. To mitigate the impact of uncertainty in a wafer fabrication process with 32 information input, Sticker et al [28] trained a DQN agent to orchestrate dispatching orders. By extending this real case, Kuhnle et al [29] modelled the wafer fabrication problem using discrete event simulation and concluded that even the simplest reward function can easily outperform fixed dispatching rules.

3. RMS Scheduling Based on Deep Q Learning

DES is a powerful tool for observing sequential jobs of an entity inside the system. DES enables decision-makers to observe the perceived performance of different scenarios, in a virtual, risk-free environment, without the need to experiment with the physical system itself. RMS scheduling has big challenges on dealing complex system state and cooperate every component inside the system. DQL is a powerful tool to find optimal choice from given state and reward. Bridging these three topics is a necessary job for operation research.

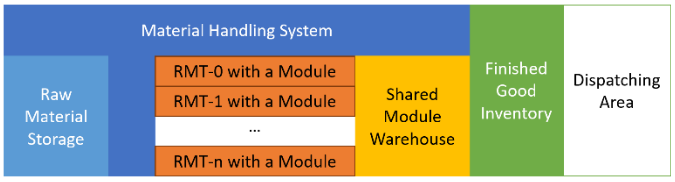

Fig. 1. A Simplified Reconfigurable Manufacturing System

A simplified version of an RMS is presented in this paper as shown in Fig 1. This dimensionless generic RMS has 4 major components. First, according to a pre-set order list, a raw material storage will have exact numbers of raw material wait for manufacturing at the beginning of a simulation episode. Second, after receiving the material dispatching order, a material handling system (MHS) will spend some time to transport the assigned material to the assigned RMT. Third component is a group of parallel RMTs with a shared module warehouse. These RMTs response on manufacturing products from material to product. Different products need different RMT module and production time. If an RMT receive a raw material current configuration cannot make, it will ask the module warehouse to send an correct module to reconfigure itself and return the original module. After an RMT finish producing a product, MHS will move the product to the last RMS component, a finished good inventory with a dispatching area. This inventory will check whether current stock fulfil the first required order. The inventory will pile up the stock until the stock level is sufficient. A trained scheduling agent should find out the policy which could transfer raw materials into products and dispatching out from RMS as soon as possible. To minimise the lead time of a given set of orders, a deep reinforcement learning model training approach based on discrete-event simulation is proposed in this section. The scheduling solution contains two major phases. First phase requires building a deep reinforcement learning agent, shown with blue blocks in Fig 2, which can learn from environment effective. Based on the observation of environment, the agent needs to pick one kind of raw material from the storage at a suitable time and send it to a free RMT inside the RMS.

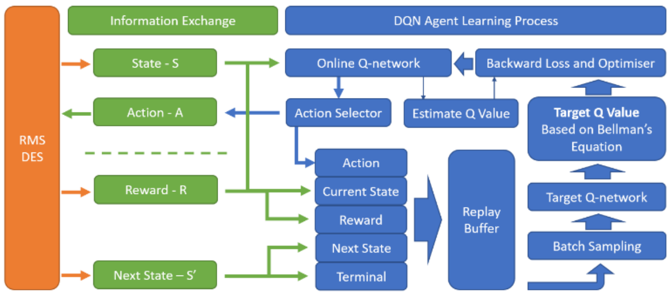

Fig. 2. Deep Q Learning Agent Training Process Flow Chart

A size-limited replay buffer records tuples which contain rewards and state-transition information feedbacked from DES. All DQN are trained from the samples from this buffer.

The second phase focuses on building a DES environment to imitate an RMS, shown in orange blocks in Fig 3, which can receive action decision, transit its state, and generate rewards. Green blocks in Figs. 2 and 3 present the information exchange.

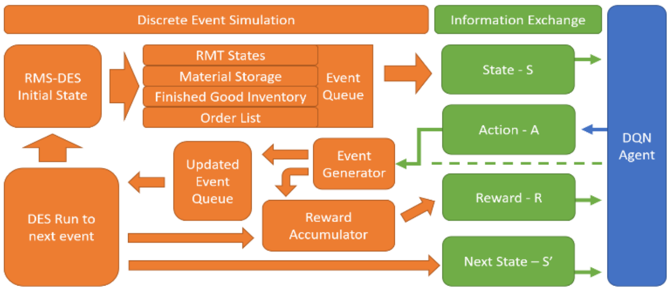

Fig. 3. Discrete Event Simulation Flow Chart

3.1. Basic DQN learning process

Basic Q learning uses Bellman's equation to describe the expected reward. The quality value of reward is presented as

$$Q(s,a) = r + \gamma argmax_{a'}\ Q(s',a') \tag{1}$$

Where r is the immediate reward for transmitting from current state, s, to next state, s'. A discount factor, $\gamma \in (0,1]$, weakens and balances the reward from future reward. Since DQL first introduction [30], there has been no significant change on the basic training process, which is shown in Table 1. The training goal of the DQN is an accurate non-linear quality function estimator, $Q(s,a)$, where s and a represent a state and an action respectively. In practice, two versions of one DQN is used for stabilising the learning process. The first one is an online neural network which is presented with a weight vector, θ, as $Q(s,a;\theta)$. The second one is the target neural network with a weight vector, θ', which is presented as $Q(s',a';\theta')$. Target neural network shares the same architecture of the online neural network but estimates the next step Q value of the next state, s', and action, a'. While the online neural network learns every training step, the target neural network keeps stable for a fixed number of iterations before the online network weight, θ, is copied to θ'. Based on this modification, Bellman's equation for DQN can be written as

$$Q(s,a;\theta) = r + \gamma max_{a'} Q(s',a';\theta') \tag{2}$$

A mean square loss function is introduced here to be gradient descent and train the online neural network from the difference between the target network. This function is written as

$$L(\theta) = \mathbb{E}[(r + \gamma \underset{a'}{argmax} Q(s',a';\theta') - Q(s,a;\theta))^2] \tag{3}$$

In this paper, $L(\theta)$ is calculated automatically by Pytorch.

For implementing the DQN algorithm, a replay memory collecting D previous experience and a deep Q-network with random weights $\boldsymbol{\theta}$ that represents the action-value function Q are initialised at the beginning of a training process. Agent will randomly choose action until the D length replay memory is fulfilled. Then scheduling agent selects and executes actions according to an ε-greedy policy based on Q. ε is the probability the action would be chosen randomly, and it will decrease until a set value as training progresses. Every training episode ends at a terminal state like run out of simulation time or all order are fulfilled.

Table 1. Basic DQL Process of a Scheduling Agent [31]

```
Input: Capacity of replay memory N, Discount Factor γ, Learning Rate α,
Episodes Limit M, Batch Sample Length L
Output: An Agent Does Optimal Scheduling
01  Initialise replay memory D with N capacity
02  Initialise an online network Q with random weights θ
03  for episode = 1 to iteration limit M
04    Observe the initial state $s_0$
05    for t = 1 to T
06      Choose an action, $a_t$, at according to Q with ε probability for a
        random action
07      execute $a_t$ on DES and get reward $r_t$ and next state $s_{t+1}$
08      Store transition ($s_t$, $a_t$, $r_t$, $s_{t+1}$) and whether it is terminal in D
09      Randomly pick L sample ($s_j$, $a_j$, $r_j$, $s_{j+1}$) from D
10      Set training target $y_j$ to
          $r_j$ if current episode terminates at step j+1
          $r_j+\gamma Q(s_{j+1}, argmax_{a'}Q(s_{j+1}, a', \theta), \theta)$ if current episode not
        terminates at step j+1
11      Perform a gradient descent step $(y_j-Q(s_{j+1}, a_j, \theta))2$
12      Decrease ε until a set minimum value
```

The basic DQL training algorithm with single neural network could sometimes overly optimistically estimating the true rewards [32]. Decoupling the action estimation and action selection can simply fix it. The main neural network would decide not only current state optimal action but also the optimal action for next state. This approach is known as Double DQN algorithm and its Bellman's equation is written as

$$\begin{cases} Q(s,a;\theta) = r + \gamma Q(s',a';\theta') \\ a' = argmax_{a'}\ Q(s',a';\theta) \end{cases} \quad (4)$$

Q value is made up two functions, V and A. The equation can be written as $Q(s,a) = V(s) + A(s,a)$.V stands for state value function of current state which tells the expected reward from current state only. A represent the advantage function which tells the difference an optimal action can bring compared to the other actions. In neural network training, V and A are unidentifiable and cannot simply sum up for Q estimation. To solve this and chasing a better agent training performance, Wang [33] introduced Dueling DQN architecture which force the highest Q-value equal to V hence highest A has to be zero. To get optimisation stability, Wang trades-off the maximum Q for an average Q so that A doesn't have to turn negative of an optimal action if A catch up the mean changing speed. The Q calculation can be re-written as

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a)\right) \quad (5)$$

3.2. States to form an observation

An input observation consists of four kinds of features, listed in Table 2, which include every RMT state, remaining raw material state, finished good stock state, and order state. State of an RMT include three elements which are installed module type, remaining time to finish reconfiguration T_{cfg}, and remaining time to finish current production T_{pdc}. The quantity of each kind of raw material still in the storage make up the remaining raw material state. The order list contains a sequence of orders. Each order consists of three elements which are product type, required quantity, and whether it has been finished or not. The finish good state describes the quantity of finished product for each kind of product.

Table 2. 4 States to Form an Observation

State	State Structure
RMT State	Module, T_{cfg}, T_{pdc}
Raw Material Stock	Quantity of each raw material
Order State	Type, Quantity, Completed or not
Finished Good Stock	Quantity of each product

3.3. Action

An optimal action, which has the highest quality value Q calculated by a forward function from neural network is the outcome from the agent for a certain state. This action is presented as $a_t = argmax_{a_t} Q(a_t|s_t,\pi)$. When training starts, the value function Q is far from accurate. Aiming on fast converging at the beginning stage, the agent should perform more arbitrary actions to explore the DES environment. A large random action choosing rate, ε, is assigned for helping agent to choose arbitrary action rather than based on a rough Q estimation. This process happens in the action selector as shown in Fig 2. With learning episode increasing, the Q-network approximate Q more accurately so ε keeps on decreasing until a pre-set minimum. The combination of all possible actions forms an action space. In this paper, there is only one agent to operate the entire RMS so that it can choose only one optimal action for a single state. For an RMS contains more than one RMT, (4) the action-space expand exponentially due to the RMT number. For example, a two-product RMS with two RMT has an action-space with 9 outputs. The optimal choice contains decisions including which kind of raw material will be send to each RMT or keep an RMT idle. The data flow of input state and output action is shown in Fig. 4. The Agent receives an observation that contains every RMT status, situation of material storage, finished good inventory, and all rest unfinished order.

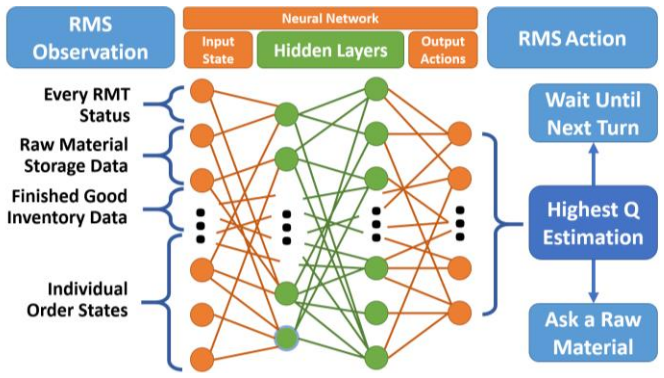

Fig. 4. Neural Network Forward Function for RMT Action

3.4. Reward and Terminal State

Designing a proper reward mechanism is critical for training an RL agent. In this paper, RMS-DES runs on discrete simulation time step. For each time step, the agent needs to decide whether to send a material for an RMT. The principle of defining reward rules is encouraging an agent keep an RMS in value-adding state instead of constant reconfiguring. While an agent implementing a logy scheduling policy, there will be no positive nor negative reward. When RMT is reconfiguring itself, it generates -50 reward per unit time. When an RMT raw material is in manufacturing, it will generate +5 reward per unit time until the process finish and the good send to the storage. One finished good in storage generate -1 reward per unit time. The agent will receive a -100 reward if it decides to send material to a busy RMT.

Table 3. Reward Rule for Unit Simulation Time

Reward Items	Reward Generating Rate
A raw material stays in a storage	0
An RMT stays in idle	0
An RMT is manufacturing a raw material	+5
An RMT is reconfiguring itself	-50
RMT is busy but agent decide to send in a material	-100
A finished good dispatched with an order	+100
All order cleared	+1,000,000

To avoid unnecessary computational power waste, there are three kinds of scenario that will end current training episode. In every simulation step, DES first updates its current state and calculate an inertial reward from previous events like reconfiguration and production. Then DES asks the agent to provide an action. This action will move DES to another state with new events and in general another instant reward which accumulates with the inertial reward. This single-step reward and new state are sent back to the agent for training while it will be accumulated to a general reward recorded inside DES to check if current simulation episode performed too bad to end. If the accumulated reward does not excess the threshold, (set as negative 2 million in all numerical cases in this paper), DES will keep running until a pre-set simulation time runs out, (set as 7200 min that equals one week), or all order are fulfilled.

4. Model Evaluation and Analysis

Three RMS cases based on the same structure are studied. An RMS can contain 1 to 3 RMTs. An RMT need to reconfigure itself to a unique configuration for manufacturing a specific product variant. The reconfiguration time for product variant 0 and 1 are identical, 300 min in all numerical cases. After an RMT reconfigures to correct form, the cycle time for producing one finished good is 1 min. All finished good will be temporally stocked until the quantity fulfils the first unfinished order. When a training episode starts, the DES environment initialises a deterministic group of sequential orders labelled with product type and quantity. One order list, shown in appendix, contains 20 individual sub-orders have been studied in this paper. This order list, *OL1*, requests 166 product 0 and 362 product 1 which will raise up to 9 reconfigurations in an RMS with single RMT when it follows FIFO. RMSs with different RMT quantity require unique DQN aggregate output layers. *RMS01* with only one RMT has 3 possible actions. *RMS02* contains 2 RMTs has 9 potential action combinations. *RMS03* with 3 RMTs has 27 possible reaction mode.

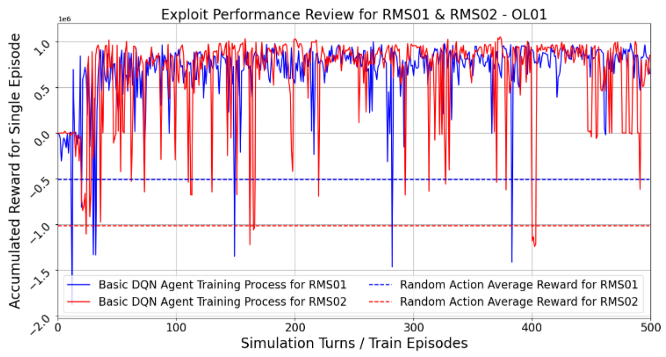

Fig. 5. Basic DQN Training Process Comparison (RMS01&02)

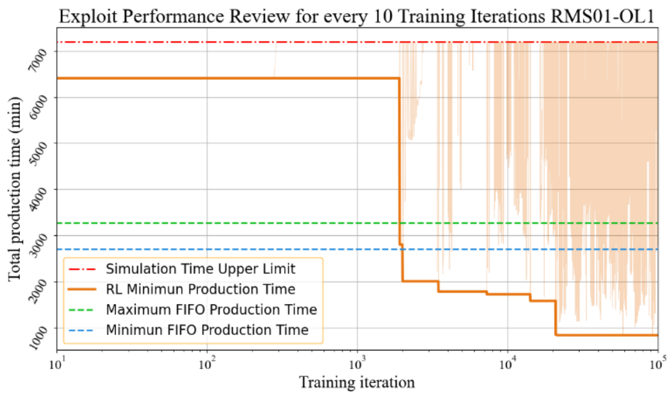

Fig. 6. Agent Performance Improving Trend

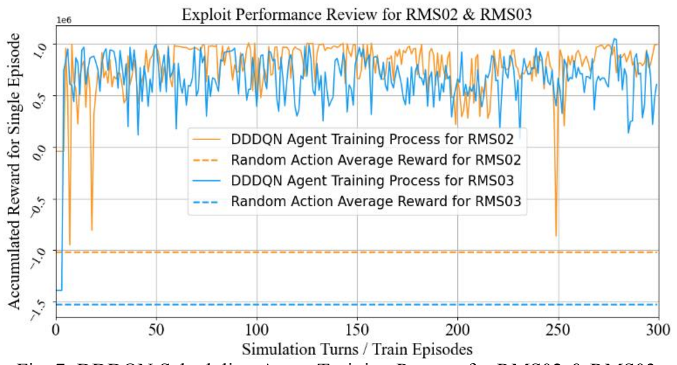

Fig. 7. DDDQN Scheduling Agent Training Process for RMS02 & RMS03

The first experience trained two basic DQN agents for two RMS setups for OL1. Agents will be trained one iteration when it gets one state-transition tuple. Every simulation turn forms one train episode which contains thousands of training iteration. After finishing one simulation turn, the agent run in exploit mode to benchmark its performance using accumulate reward. Two solid lines in Fig 5 present the benchmark trends of two agents of RMS01 and RMS02. Dashed lines are the average accumulated rewards when RMS01 and RMS02 act randomly. Fig. 5 also illustrate that both agents converged quickly after only around 30 turns. However, increasing training episodes did not bring more advantages to agents. Some episodes' ill performance even worse than complete random actions.

When an RMS perform in FIFO mode, raw material will dispatch to every free RMT only according to the order sequence only regardless the RMT current configuration. To overcome this shortcoming, a far-sighted agent needs to combine orders with the same variant together to reduce total reconfiguration action hence minimising the total production time. Fig. 6 shows a DDDQN agent training process in a One-RMT RMS. Due to different initial RMT configuration, FIFO can finish the OL1 around 3000 mins. Start from a poor performance level, the agent quickly outperforms FIFO after only training iterations. The agent achieved its optimal strategy at around 20 thousand iterations.

Aiming on improving the training stability, two Double DQN agents with duelling architecture are set for RMS02 and RMS03. Fig. 7 shows that DDDQN agent perform more stable after converging compares to the basic DQN agent.

5. Conclusions

DQN scheduling agents showed great performance advantage potential in this paper. Agents out-perform a traditional dispatch rule, first-in-first-out, in very limited simulation turns. The training results show that all kinds of DQN agent can quickly converge to an above average level. Dueling Double DQN agents shows an advanced stability as its inventor suggested. However, even if the converging capability and stability of DDDQN is much premium compares to the basic DQN, it is clear that an agent can perform way from optimal occasionally regardless the increasing training episodes. Keeping on optimising the agent stability with advanced techniques, such as priority replay memory, bagging strategy is necessary. All agent converges in several episode suggest the over-simplified RMS cannot full unveil the agent potential. The only randomness inside the environment is the initial configuration of every RMT. Material handling system with fluctuated deliver time, random breakdown on RMT, limited number of modules, material resource, products with multi manufacturing processes, and dynamic order lists with task inserting should be considered in a future model.

Appendix A. Order List

Order List 1 - OL1: 166 Product 0 and 362 Product 1

Order #	Variant	Quantity	Order #	Variant	Quantity
0	0	22	10	0	24
1	1	27	11	1	19
2	1	47	12	0	19
3	1	31	13	1	22
4	0	17	14	1	43
5	1	20	15	1	20
6	1	24	16	0	33
7	1	24	17	0	21
8	1	40	18	1	35
9	0	30	19	1	10

References

1. Koren Y, Heisel U, Jovane F, et al (1999) Reconfigurable Manufacturing Systems. CIRP Ann 48:527–540.
2. Mittal KK, Jain PK (2014) An Overview of Performance Measures in Reconfigurable Manufacturing System. Procedia Eng 69:1125–1129.
3. Moghaddam SK, Houshmand M, Saitou K, Fatahi Valilai O (2020) Configuration design of scalable reconfigurable manufacturing systems for part family. Int J Prod Res 58:2974–2996.
4. Koren Y, Shpitalni M (2010) Design of reconfigurable manufacturing systems. J Manuf Syst 29:130–141.
5. Napoleone A, Pozzetti A, Macchi M (2018) Core Characteristics of Reconfigurability and their Influencing Elements.
6. Tang J, Emmanouilidis C, Salonitis K (2020) Reconfigurable Manufacturing Systems Characteristics in Digital Twin Context. IFAC-PapersOnLine 53:10585–10590.
7. Luo S (2020) Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Appl Soft Comput J 91:106208.
8. Bakakeu J, Tolksdorf S, Bauer J, et al (2018) An Artificial Intelligence Approach for Online Optimization of Flexible Manufacturing Systems. Appl Mech Mater 882:96–108.
9. Zhang Z, Zheng L, Weng MX (2007) Dynamic parallel machine scheduling with mean weighted tardiness objective by Q-Learning. Int J Adv Manuf Technol 34:968–980.
10. Watkins CJCH, Dayan P (1992) Q-learning. Mach Learn 8:279–292
11. Mnih V, Kavukcuoglu K, Silver D, et al (2015) Human-level control through deep reinforcement learning. Nature 518:529–533.
12. Mnih V, Kavukcuoglu K, Silver D, et al (2013) Playing Atari with Deep Reinforcement Learning. 1–9
13. Aydin ME, Öztemel E (2000) Dynamic job-shop scheduling using reinforcement learning agents. Rob Auton Syst 33:169–178.
14. Ahmadian MM, Khatami M, Salehipour A, Cheng TCE (2021) Four decades of research on the open-shop scheduling problem to minimize the makespan. Eur J Oper Res.
15. Xu W, Guo S (2019) A multi-objective and multi-dimensional optimization scheduling method using a hybrid evolutionary algorithms with a sectional encoding mode. Sustain 11:.
16. Xiao H, Wu X, Zeng Y, Zhai J (2020) A CEGA-Based Optimization Approach for Integrated Designing of a Unidirectional Guide-Path Network and Scheduling of AGVs. Math Probl Eng 2020:.
17. Carli R, Digiesi S, Dotoli M, Facchini F (2020) A control strategy for smart energy charging of warehouse material handling equipment. Procedia Manuf 42:503–510.

18. Guan J, Lin G, Feng H Bin, Ruan ZQ (2020) A decomposition-based algorithm for the double row layout problem. Appl Math Model 77
19. Palombarini JA, Martinez EC (2019) Closed-loop rescheduling using deep reinforcement learning. IFAC-PapersOnLine 52:231–236.
20. He Y, Wu G, Chen Y, Pedrycz W (2021) A Two-stage Framework and Reinforcement Learning-based Optimization Algorithms for Complex Scheduling Problems. 1–11.
21. Huang J, Chang Q, Arinez J (2020) Deep reinforcement learning based preventive maintenance policy for serial production lines. Expert Syst Appl 160:113701.
22. Han W, Guo F, Su X (2019) A reinforcement learning method for a hybrid flow-shop scheduling problem. Algorithms 12.
23. Souier M, Dahane M, Maliki F (2019) An NSGA-II-based multiobjective approach for real-time routing selection in a flexible manufacturing system under uncertainty and reliability constraints. Int J Adv Manuf Technol 100:2813–2829.
24. Barto RSS and AG (1998) Introduction to Reinforcement Learning
25. Creighton DC (2002) Proceedings of the 2002 winter simulation conference - Volume 2. Winter Simul Conf Proc 2:1945–1950.
26. Gabel T, Riedmiller M (2012) Distributed policy search reinforcement learning for job-shop scheduling tasks. Int J Prod Res 50:41–61.
27. Waschneck B, Reichstaller A, Belzner L, et al (2018) Optimization of global production scheduling with deep reinforcement learning. Procedia CIRP 72:1264–1269.
28. Stricker N, Kuhnle A, Sturm R, Friess S (2018) Reinforcement learning for adaptive order dispatching in the semiconductor industry. CIRP Ann 67:511–514.
29. Kuhnle A, Kaiser JP, Theiß F, et al (2021) Designing an adaptive production control system using reinforcement learning. J Intell Manuf 32:855–876.
30. Mnih V, Kavukcuoglu K, Silver D, et al (2013) Playing Atari with Deep Reinforcement Learning. 1–9.
31. Sutton RS, Barto AG (2018) Reinforcement Learning: An Introduction.
32. Hasselt H van, Guez A, Silver D (2016) Deep Reinforcement Learning with Double Q-Learning. Proc Thirtieth AAAI Conf Artif Intell 30:7.
33. Wang Z, Schaul T, Hessel M, et al (2016) Dueling Network Architectures for Deep Reinforcement Learning. 33rd Int Conf Mach Learn ICML 2016 4:2939–2947.