Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 56-2 (2023) 11082–11087

Multi-objective Reconfigurable Manufacturing System Scheduling Optimisation: A Deep Reinforcement Learning Approach

Jiecheng Tang. Yousef Haddad. John Patsavellas. Konstantinos Salonitis.

Sustainable Manufacturing Systems Centre, School of Aerospace, Transport and Manufacturing, Cranfield University, Bedford, MK43 0AL, UK

(e-mail: jiecheng; yousef.haddad; john.patsavellas; k.salonitis@cranfield.ac.uk,)

Abstract: Rapid product design updates, unstable supply chains, and erratic demand phenomena are challenging current production modes. Reconfigurable manufacturing systems (RMS) aim to provide a cost-effective solution for responding to these challenges. However, given their complex adjustable nature, RMSs cannot fully unlock their potential by applying old-fashion fixed dispatching rules. Reinforcement learning (RL) algorithms offer a useful approach for finding optimal solutions in such complex systems. This paper presents a framework to train a scheduling agent based on a proximal policy optimisation (PPO) algorithm. The results of a numerical case study that implemented the framework on a simplified RMS model, suggest a good level of robustness and reveal areas of unpredictable behaviour that could be the focus of further research.

Copyright © 2023 The Authors. This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0/)

Keywords: Manufacturing plant control, Reconfigurable manufacturing system, reinforcement learning, scheduling, proximal policy optimisation

1. INTRODUCTION

In contrast to the manufacturing paradigms of high-throughput dedicated production lines and wide functionality flexible manufacturing systems, reconfigurable manufacturing systems (RMS) target medium-volume-medium-variant mixed order production. Since their conceptual introduction by Koren et al., (1999), new technologies that can help to build practical RMS became mature and accessible. An RMS provides capabilities around 6 core characteristics that include modularity, integrability, diagnosability, scalability, customisability and convertibility. Aiming on high customisation, a digitalised RMS highly relies on information exchange to secure its more fundamental characteristics of modularity and diagnosability (Tang et al., 2020). However, whilst vital for RMS efficacy, extra information brings extra challenges on an RMS. Whilst the problem of handling fluctuating demand promoted the birth of RMS, their scheduling can be considered as a special branch of the flexible job-shop scheduling problem (FJSP). In such a problem, the conventional fixed dispatching policies like first-in-first-out cannot fully release the potential of an RMS. Meta-heuristics like genetic algorithms (Dou et al., 2018, 2019, 2021) were considered good approaches to the challenge of finding an optimal scheduling solution. In recent years, reinforcement learning (RL) has revealed great potential in finding a credible scheduling policy (Tang & Salonitis, 2021). RL has gained achievements in many fields including gaming (Silver et al., 2016, 2017; Vinyals et al., 2017), biotechnology (Jumper et al., 2021), nuclear fusion reaction control (Degrave et al., 2022), and even mathematics (Fawzi et al., 2022). Some of these problems share a common pattern of optimising one or a limited number of measures in real-time from a predictable environment with complicated input. The FJSP of RMS fit this pattern as well. The unpredictable demand makes a roomy observation space while the way how an RMS operating is predictable. Although research of finding an optimal schedule for an RMS in real-time is not fully developed, there are considerable amount of papers regarding RL methods to solve FJSPs even before the RMS paradigm was introduced (Zhang & Dietterich, 1995a, 1995b).

There are two major approaches to solve FJSP by RL algorithms. The first one is on-policy methods developed from the principle of RL direct interaction. On-policy algorithms upgrade their activity policies after every interaction with a given environment (Sutton & Barto, 2018). Such algorithms tend to suffer from low sample efficiency prompting researchers to develop off-policy methods represented by deep Q networks (DQN) that can store data firstly in a buffer and then train the policy later. The DQN method was deployed to improve a wafer fabrication scheduling system on equipment utilisation by 8% and decrease lead time by 7 minutes (Stricker et al., 2018). Waschneck et al., (2016, 2018b, 2018a) modelled a semiconductor factory through discrete-event simulation (DES) and developed a suboptimal scheduling policy with DQN methods. Although DQN methods appear to have some promise their training process can sometimes fall into computational traps without recovery. This phenomenon forced researchers to reconsider on-policy algorithms and developed Proximal Policy Optimisation (PPO) (Schulman et al., 2017). PPO provided promising results in scheduling problems. For example, Rummukainen & Nurminen, (2019) formulated a stochastic economic lot scheduling problem into a semi-Markov setting. In their 3×10^9 state space environment, PPO provided a promising control policy after 11×10^7 training iterations.

2405-8963 Copyright © 2023 The Authors. This is an open access article under the CC BY-NC-ND license.
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2023.10.814

Aiming on generate a real-time updating schedule of RMS, this paper proposes a framework to train a reinforcement learning-based scheduler. The following sections of the paper are organised as follows: section two presents the general structure of the proposed framework to train the scheduling agent. Similar to all other reinforcement learning projects, this general framework contains three major parts, a Markov decision process environment to mimic an RMS, a reward function which express the optimising goals, and a job releasing schedule agent which contains the policy. Section three presents a numerical case study about verifying the framework by training a scheduling agent through the PPO algorithm for a simplified DES-based RMS model. Further analysis presenting processing results follow in section three. The final section concludes with the advantages and disadvantages of the proposed framework and summarises the prospects for future research.

2. A REINFORCEMENT LEARNING SCHEDULER TRAINING FRAMEWORK

In a previous study, Tang et al., (2022) addressed the multi-agent RMS scheduling problem, proposing a multi-agent RMS scheduling agent based on an off-line reinforcement learning algorithm called Dueling Double Deep Q-Network (DDDQN) with a Prioritised Experience Replay (PER). Although this framework was shown to be feasible, the converging speed was slow. To overcome this limitation, this paper introduces a newly designed framework focused on reconfigurable machine tools (RMT), and employs a cutting-edge reinforcement learning algorithm called Proximal Policy optimisation (PPO) to train an intelligent scheduler within reasonable computational time.

Similar thus to the RMS model proposed by Tang et al. (2022), this paper presents a simplified dimensionless RMS which can produce limited variants of a product. Such an RMS consists of a fixed number of RMTs. These uniform RMTs can produce different products with matched modules which need time to be reconfigured. The goal of such an RMS is to fulfil a constantly updating order list. This order list contains a certain number of individual orders with each order consisting of a random quantity of jobs. The deadlines are set as a function of the order size; i.e. the bigger the order the longer the deadline. Once the deadlines are set, then they are considered rigid and non-negotiable. If an order missed the deadline due to no sufficient inventory, there will be no extension or delay to fulfil the demand. In that case a new order will be dispatched automatically to the RMS stochastically regardless of whether it was success or not. This is described schematically in Fig. 1.

The scheduling agent is designed to generate schedules for every individual RMT. These maintenance-free RMTs would equip a random module when initialising the RMS. When the initialisation ends, the number of RMTs cannot be changed. The agent generates a schedule by sending two types of action commands to RMTs at every decision-making instance. The first type of action is “stay rest”. In such a case, the RMT will stay idle and generate neutral reward feedback for training.

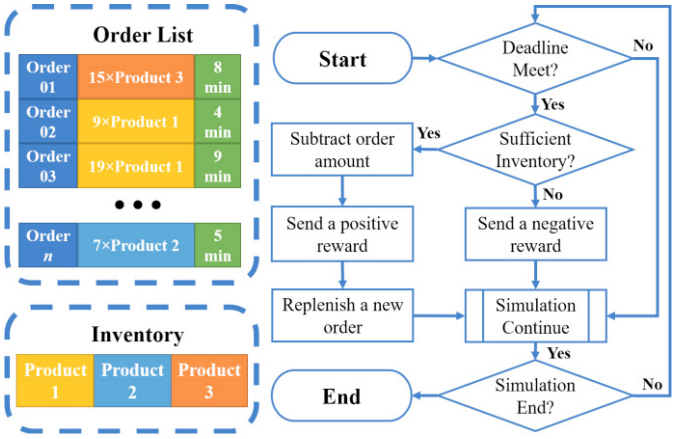

Fig. 1. A stochastically updating order list with 3 products

The second type of action is to produce a certain kind of product. In this circumstance, if an available RMT equips the proper module to produce the required product, this RMT will produce one product and send it to the inventory (if produced before the deadline). If the module that RMT is equipped with cannot follow the action order, the RMT will initiate reconfiguration of itself with a proper module (which will require a set amount of time) and then produce the required product and send it to the inventory. For example, if an RMT equips Module 1 which can produce Product 1 while the scheduler asks it to produce Product 1, the RMT will produce one Product 1 with a certain lead-time. If the scheduler asks an RMT to produce Product 2 while the RMT is equipped with Module 1, the RMT will spend some time to reconfigure itself with Module 2 and then produce one Product 2 with a certain lead-time. All products will be collected into a finished-good inventory before being dispatched to customers. To formalise the procedure to enhance applicability and transferability, Fig. 2 presents the flowchart below.

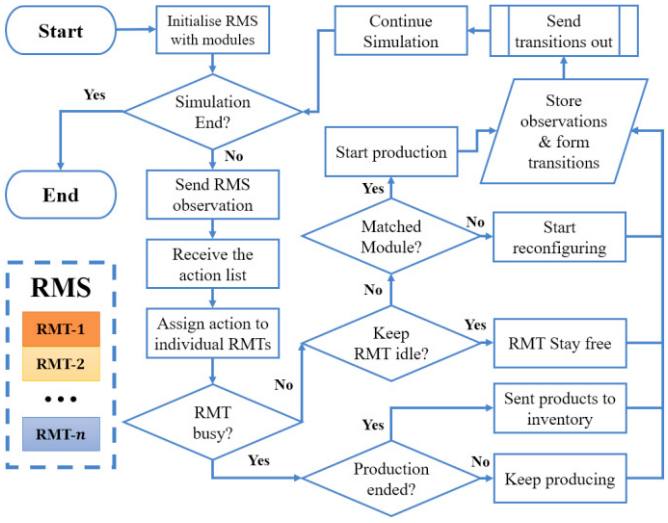

Fig. 2. RMS Simulation Flowchart

For the agent to be able to learn how to make decisions, a reward policy is set. Considering thus the intelligent scheduling policy as a function for reinforcement learning algorithm to solve, the optimal solution for such functions is fulfilling as much as order while keep the inventory level as low as possible. To solve such a function properly under RL content, the reward function framework shown in Table 1 is proposed.

Table 1. Reward function category

Scenario	Reward
Gain a product	Small positive
Keep a product inventory	Small negative
Fulfil an order	Huge positive
Miss an order	Medium negative
Reconfigure an RMT	None
Keep an RMT rest	None

The idea behind the proposed reward framework is to augment the signal from missing or delivering an order, while providing to the agent a relative dense reward signal from the inventory during training. For increasing the compatibility of the environment, all forementioned simulation features of an RMS are following the OpenAI-gym interface developing rule.

Once the two out of the three major components, namely environment and reward, of the RL project are ready, the agent needs to be built. This paper involves the development of a resilient framework for the training of the agent to take place. Unlike the supervised learning that is based in a steady and clear labelled data set, stochastic gradient descent (SGD) cannot guarantee a decent local optimum in RL. The training data thus for RL are generated being highly dependent on current policy and "burnt after training". The constantly changing data distribution of observation and rewards lead to instability during agent training process. RL training is also highly sensitive to hyperparameter tuning such as initialisation (Schulman et al., 2017). One slight change on policy can have a great impact on the performance. PPO algorithm is developed for minimising the cost function by balancing among small policy update, ease of implementation, sample complexity, and ease of hyperparameter tuning. PPO is a policy gradient method which belong to the on-policy category. It is different from off-policy algorithms like DQN, as the learning happens with exploration and is able to learn from recorded data. On-policy agents always learn directly from whatever they encounter in the environment, following their own policies and cannot store past experiences in a replay buffer. Additionally, a batch of experience will be discarded immediately once it has been used to do a gradient update for a policy.

Policy Gradient (PG) methods use samples of the interactions only once, which mean they are significantly less efficient. The general policy optimisation methods define the policy gradient loss, $L^{PG}(\theta) = \hat{E}_t\left[log\, \pi_\theta(a_t|s_t)\hat{\mathrm{A}}_t\right]$, as the expectation over the probabilities of the policy actions, times an advantage function estimation. The current policy, π_θ, with the policy parameter θ takes the state of system, s_t, as input and output an action, a_t. An advantage, $\hat{\mathrm{A}}_t$, is an estimation of a relative value of the selected action in s_t. To calculate $\hat{\mathrm{A}}_t$, the sum of discounted rewards and a value-function estimation are needed. The discounted reward is a weighted sum of all the rewards happen after taking current action which can be presented as $E = \sum_{k=0}^{\infty} \gamma^k r_t + k$. These advantages are not an estimation as their value are calculated after the entire episode sequence has been collected. The value function, which is the second part of the advantage, is defined as the estimation of discounted reward or final reward from this step onward. In PPO, the value function is presented as a neural network. By using the state and discounted reward records, the value function network can be updated similar to a supervised learning problem. In this case, the advantage estimation can be presented as discounted rewards, a certain number, subtract the value estimation, a neural network estimation. The advantage can then instruct the agent whether the actions took return a better or worse result than expected. According to that, the probability of an agent choosing such an action can be increased or decreased respectively. To train the policy, π_θ, SGD are used to minimise $L^{PG}(\theta)$. However, if an agent keeps running gradient descent on the same batch of collected rollouts, the neural network would be updated to a far outside the reasonable range as the advantage itself is a noisy estimation.

PPO brought the "Trust Region" concept from Trust Region Policy Optimization (TROP), another iterative policy optimisation approach, that never update the policy too far away from the old one. By replacing the log operation with a division by the old policy, the objective function can be presented as a progressive empirical expectation $L(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{\mathrm{A}}_t\right]$. Define the probability ratio between the new and old policy output as a variable $r_t(\theta) = \pi_\theta(a_t|s_t) \cdot \pi_{\theta_{old}}(a_t|s_t)^{-1}$. In such a case, $r(\theta_{old})$ is equal to 1 and $L(\theta)$ can be regarded as "surrogate" objective (Schulman et al., 2015). While $r_t(\theta)$ is larger than one, if the sampled action is more likely to be chosen in the new policy and rest between 0 and 1 if such action gets less chance before the last gradient update. The central optimisation objective of PPO can be written in relative conservative form as below.

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{\mathrm{A}}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{\mathrm{A}}_t)]$$

An expectation operator takes the minimum of either the normal policy gradients objective $r_t(\theta)\hat{\mathrm{A}}_t$, which push the policy to yield a high positive advantage over the baseline, or a truncated version of $r_t(\theta)$ with an exploration hyperparameter ε which usually between 0.1 and 0.2, which limits the update step in case it goes too far based on a single noisy estimation. Considering the policy and value function share parameters, a squared-error loss function $L_t^{VF}(\theta)$ is involved. An entropy bonus $S[\pi_\theta](s_t)$ is used ensure keeping agent exploring and the final training objective can now be written as below.

$$L_t^{PPO}(\theta) = \hat{E}_t[L^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

Combining the simulation environment and agent training process, the framework to training a scheduling agent can be present as pseudo code below.

Clipped PPO-based Scheduling Agent Training Scheme

Randomly initialise N parallel environments each with a distinct order list, an RMS containing m RMTs, and an initially empty inventory that can store different kinds of products
Initialise a job-releasing agent with a random scheduling policy π with parameter θ
for iteration = 1 **to** limit M **do**
  for actor = 1 **to** N **do**
    for time t = 1 **to** simulation limit T **do**
      Form an observation s_t from all known orders and their deadlines, inventory levels, and RMT states
      Request an action list a_t which contains individual commands for every RMT
      Simulate 1 timestep according to a_t
      Form another observation s_{t+1} based on the same method as s_t
      Calculate the reward r_t based on s_t, a_t, and s_{t+1}
      Form a transition $\{s_{t+1}, r_t | s_t, a_t\}$
    end for
    Calculate advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$ for all transitions and store to a rollout buffer
  end for
  for epoch = 1 **to** K **do**
    Optimise the loss objective $L^{PPO}(\theta)$ by sampling a minibatch of size $\leq NT$
  end for
end for

3. A NUMERICAL CASE STUDY

To verify the proposed RL framework, one numerical case study is designed and presented in this section. The following section will firstly describe the parameter setting for initialising an RMS model, the reward function and the scheduling agent. The training and performance changing processes are presented and discussed progressively.

This case study used a simplified discrete-event simulation (DES) model to describe an RMS. DES provides a feasible framework to transfer the real production process into a Markov Decision Process (MDP). The critical feature of DES which is driving the model by events, helps the RL agent identify the decision points.

The RMS-DES model in this case contains three identical RMTs. All RMTs can reconfigure themselves with three different modules for three different products. The production cycle times for these products are equal (set at 1 minute) and the reconfiguration time for all modules is 3 minutes. When initialising the model, all three RMTs would be assigned to a random module. To focus on reconfiguration and production only, all material and finished goods transportation time is considered negligible. The scheduling agent only need to decide what action every RMT should take at every timestep in a total of 960 minutes simulation.

The stochastic order generator sets the relevant model parameters such as the order type and size. Every single order thus is characterized by the product type (randomly setting one out of three) and the batch size (that is defined from a uniform random distribution ranging between 5 and 20 parts). The deadline for completing the order is also randomly set, taking however into consideration the batch size. This assumption allows for the RMS in most of the orders to have just enough time, to reconfigure RMTs, complete the production and at the same time indirectly penalize the agent if it misses potential gains. There will be always 5 orders in queue for the scheduling agent to plan ahead.

As the general performance signal for the RL agent, the reward function of this case study is set as shown in Table 2.

Table 2. Case Study Reward function

Scenario	Reward per timestep
Gain a product	5
Keep a product inventory	-1
Fulfil an order	50 × Required quantity
Miss an order	1 ×Required quantity
Reconfigure an RMT	0
Keep an RMT rest	0

At the last step, a special reward is set for summarising the delivered orders. This negative reward value equals to an accumulative sum of the remaining inventory. For example, if there are 5 products, regardless the product variant after the first order was delivered, this cumulative sum equals to 5. Then, if there are 3 products remained in the inventory after the second order is delivered, this sum adds up to 8.

The policy and value networks of the scheduling agent trained for this case study have similar structure. Both neural networks start with three dense layers where each layer contains 8,192 neurons. The value network has another two dense layers, 1,024 and 256 neurons respective, before output layer while the policy layer contains only one 1,024-neuron dense layer. The learning rate is set as 0.00005 as the training with 0.0001 failed. The discount rate set as 0.9996 so that the expected values of first action can still worth roughly to 70% at the last step. The clip range for both networks is set as 0.1 and maximum value for the gradient clipping set 0.5. Entropy coefficient set 0.003 for forcing the agent exploring unknown. When optimising the surrogate loss, the agent runs for 36 epochs.

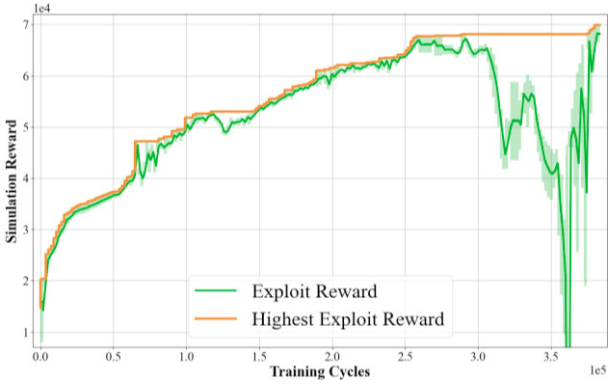

Fig. 3. Exploit Performance Reward Record

Finally, the training process runs on a vectorised environment with 36 sub-environments. Fig. 3 below shows the

performance of the training process in a deterministic way. Every training cycle collects 1,036,800 transitions and trains the two networks by using 34,560 as the mini-batch size. The fade green area is the individual performance check after every training cycle. The solid green line is the rolling mean of these, exploiting reward every 1,800 cycles. The solid orange line represents the highest reward the agent ever achieved during training.

The reward trend observed in this case study is consistent with that of other RL studies (Sutton & Barto, 2018), (Schulman et al., 2015), (Schulman et al., 2017), but the significant surge towards the end demands further investigation. As specified in the introduction, the proposed RL training framework seeks to train a scheduling agent that delivers the maximum number of orders while ensuring a reasonable inventory level. To scrutinise this trade-off, three performance comparisons were conducted. The first one compared the delivered orders versus the missed ones in a single simulation. Subsequently, the inventory control performance is evaluated in two ways. Assessing the effectiveness of RMT reconfiguration came at last.

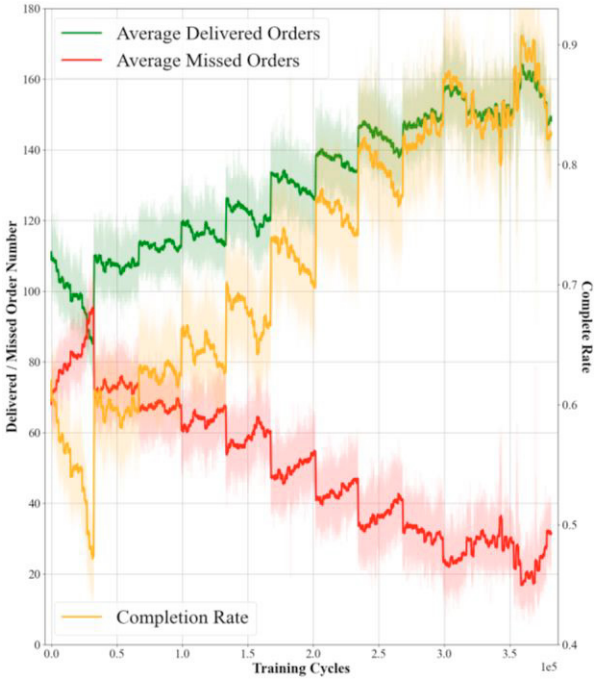

Fig. 4. Order Completion Record

Fig. 4 displays three correlated indices for the RMS order delivery. The green solid line represents the average number of orders that the agent can deliver across 36 different environments. The individual results are depicted in the faded green area. The solid red line and faded red area indicate the average and individual numbers of missed orders, respectively. The orange solid line and faded orange area reflect the average and individual order completion rates, combining both delivered and missed orders. The order completion rate shows a steady upward trend until it reaches the theoretical limit, which signifies that the primary objective of this project has been achieved. However, it is important to note that Fig. 4 alone does not provide enough evidence to explain the reward jump and recovery in Fig. 3.

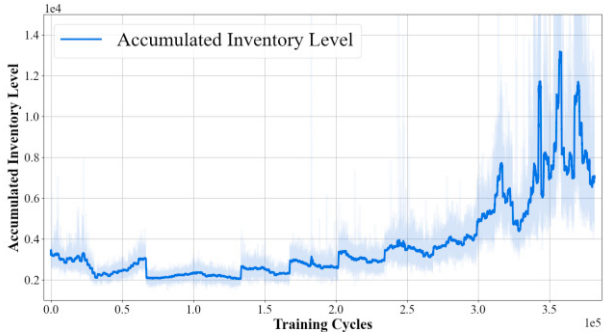

Fig. 5. Accumulated Inventory Record

To gain deeper insight into the observed phenomena, Fig. 5 is presented to illustrates the changing trends in the cumulative inventory level as training progresses. The analysis shows that the average cumulative inventory level remains approximately constant at 3,000 until a sudden drop in reward is observed. Fig. 6 presents the trend of the average inventory level per order by dividing the average cumulative inventory by the average number of delivered orders. This approach helps exploring the spread of individual orders. The plot indicates that the agent attempted to increase the stock level, likely in an attempt to improve order completion rates given the high instant reward associated with such efforts.

Fig. 6. Average Inventory Level for Individual Orders

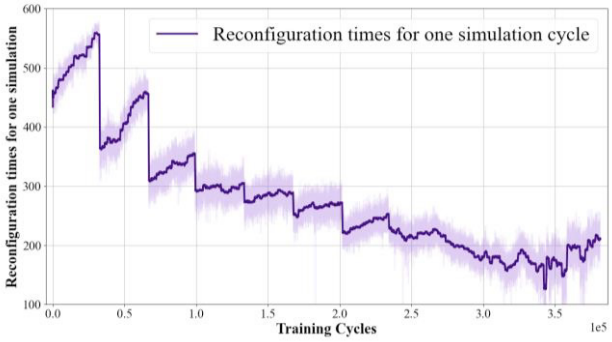

Fig. 7. Total Reconfiguration Times Record

In Fig. 7 above, the trend of the average sum of total reconfiguration actions is plotted. The trend suggests that reducing reconfiguration actions is the major driver of improving order completion rate. While this driver starts losing its momentum, the agent has to seek another way as to achieve higher reward.

4. CONCLUSION AND FUTURE WORKS

The agent shows astonishing resilience on fulfilling fluctuate demands. Emphasising the major optimising target can help the agent overcome some frustrating performance drop.

Although PPO is developed to fight against some disastrous situations like one policy update pushes the policy network into a region of the parameter space where next batch of data under a very poor policy causing it to never recover again, some training records still suggested that training could fall into traps with a bad hyperparameter setting. The other major problem showed on case study, similar to other RL projects, is the time-consuming training process. An RMS may involve new modules or new RMTs to expand its capability or the other way around. In such circumstances, week-long training time would significantly influence the deploying process.

This research focused on improving the convertibility of an RMS by training a smart schedule policy. However, the number of RMT is fixed which means current setup lacks consideration of diagnosability, such as breakdowns, nor scalability, adding or removing machines or modules from systems, of an RMS. All possible improvement aiming on the six core characteristics of RMS can bring observation space and action space change. These changes will challenge the scheduling agent on transfer learning (Taylor & Stone, 2009) and practical deployment.

REFERENCES

Degrave, J., Felici, and Other, (2022). *Magnetic control of tokamak plasmas through deep reinforcement learning.* Nature, 602(7897), 414–419.

Dou, J., Li, J and Other, (2021). A multi-objective particle swarm optimisation for integrated configuration design and scheduling in reconfigurable manufacturing system. International Journal of Production Research, 59(13), 3975–3995.

Dou, J., Zhao, X., and Other, (2018). *An Improved Genetic Algorithm for Optimization of Operation Sequencing.* 2018 IEEE International Conference on Mechatronics and Automation (ICMA), 695–700.

Dou, J., Zhao, X., and Other, (2019). *Robust Optimization Models of Integrated Configuration Design and Scheduling for Reconfigurable Flowline.* 2019 IEEE International Conference on Mechatronics and Automation (ICMA), 70–75.

Fawzi, A., Balog, M., and Other, (2022). *Discovering faster matrix multiplication algorithms with reinforcement learning.* Nature, 610(7930), 47–53.

Jumper, J., Evans, R., and Other, (2021). *Highly accurate protein structure prediction with AlphaFold.* Nature, 596(7873), 583–589.

Koren, Y., Heisel, U., and Other, (1999). *Reconfigurable Manufacturing Systems.* CIRP Annals, 48(2), 527–540.

Rummukainen, H., and Nurminen, J. K. (2019). *Practical Reinforcement Learning -Experiences in Lot Scheduling Application.* IFAC-PapersOnLine, 52(13), 1415–1420.

Schulman, J., Levine, S., and Other, (2015). *Trust Region Policy Optimization.* In F. Bach & D. Blei (Eds.), Proceedings of the 32nd International Conference on Machine Learning (Vol. 37, pp. 1889–1897). PMLR.

Schulman, J., Wolski, F., and Other, (2017). *Proximal Policy Optimization Algorithms.* CoRR, abs/1707.0.

Silver, D., Huang, A., and Other, (2016). *Mastering the game of Go with deep neural networks and tree search.* Nature, 529(7587), 484–489.

Silver, D., Schrittwieser, J., and Other, (2017). *Mastering the game of Go without human knowledge.* Nature, 550(7676), 354–359.

Stricker, N., Kuhnle, A., and Other, (2018). *Reinforcement learning for adaptive order dispatching in the semiconductor industry.* CIRP Annals, 67(1), 511–514.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning, second edition: An Introduction. MIT Press.

Tang, J., Emmanouilidis, C., and Other, (2020). *Reconfigurable Manufacturing Systems Characteristics in Digital Twin Context.* IFAC-PapersOnLine, 53(2), 10585–10590.

Tang, J., Haddad, Y., and Other, (2022). Reconfigurable manufacturing system scheduling: a deep reinforcement learning approach. Procedia CIRP, 107.

Tang, J., & Salonitis, K. (2021). A Deep Reinforcement Learning Based Scheduling Policy for Reconfigurable Manufacturing Systems. Procedia CIRP, 103, 1–7.

Taylor, M. E., & Stone, P. (2009). *Transfer Learning for Reinforcement Learning Domains: A Survey.* In Journal of Machine Learning Research (Vol. 10).

Vinyals, O., Ewalds, and Other, (2017). *StarCraft II: A New Challenge for Reinforcement Learning.*

Waschneck, B., Altenmüller, T., and Other, (2016). Production Scheduling in Complex Job Shops from an Industry 4.0 Perspective: A Review and Challenges in the Semiconductor Industry. SAMI@ IKNOW, 1–12.

Waschneck, B., Reichstaller, and Other, (2018a). *Deep reinforcement learning for semiconductor production scheduling.* 2018 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC), 301–306.

Waschneck, B., Reichstaller, and Other, (2018b). *Optimization of global production scheduling with deep reinforcement learning.* Procedia CIRP, 72, 1264–1269.

Zhang, W., & Dietterich, T. (1995a). *High-performance job-shop scheduling with a time-delay TD (λ) network.* Advances in Neural Information Processing Systems, 8.

Zhang, W., & Dietterich, T. (1995b). *A Reinforcement Learning Approach to Job-shop Scheduling.* 1995 International Joint Conference on Artificial Intelligence, 1114–1120.

https://dspace.org/

Multi-objective reconfigurable manufacturing system scheduling optimisation: a deep reinforcement learning approach

Tang, Jiecheng; Haddad, Yousef; Patsavellas, John; Salonitis, Konstantinos

2023-11-22
Elsevier
https://doi.org/10.1016/j.ifacol.2023.10.814
Tang J, Haddad Y, Patsavellas J, Salonitis K. (2023) Multi-objective reconfigurable manufacturing system scheduling optimisation: a deep reinforcement learning approach. IFAC-PapersOnLine, Volume 56, Issue 2, pp. 11082-11087

Downloaded from a DSpace repository. https://dspace.org/