



Jiecheng Tang

A Complete Reinforcement Learning based Framework for
Reconfigurable Manufacturing System Scheduling

School of Aerospace, Transport and Manufacturing
PhD in Manufacturing

PhD Thesis
Academic Year: 2018 - 2022

Supervisor: Professor Konstantinos Salonitis
Associate Supervisor: Dr. Christos Emmanouilidis
November 2022



School of Aerospace, Transport and Manufacturing
PhD in Manufacturing

PhD

Academic Year 2018 - 2022

Jiecheng Tang

A Complete Reinforcement Learning based Framework for
Reconfigurable Manufacturing System Scheduling

Supervisor: Professor Konstantinos Salonitis
Associate Supervisor: Dr. Christos Emmanouilidis
2018 - 2022

© Cranfield University 2022. All rights reserved. No part of this
publication may be reproduced without the written permission of the
copyright owner.

ABSTRACT

Since the last decade of the 20th century, a new kind of manufacturing system paradigm known as the reconfigurable manufacturing system (RMS) has been emerging. The purpose of an RMS is to offer a balanced solution that can swiftly respond to volatile global markets with fluctuating product demand. It achieves this by combining the high throughput of conventional dedicated manufacturing lines (DMLs) with the flexibility of flexible manufacturing systems (FMSs). To instigate the market uncertainty, RMS possesses six core characteristics, namely modularity, integrability, convertibility, scalability, diagnosability, and customisation. These core characteristics are becoming more available with the development of Industry 4.0 technologies. Simulation on digital twins is one of the compelling approach that help RMSs check their status in real time. However, the extended data flow challenges the traditional rule-based scheduling policies and urges a flexible approach to replace ill-suited approaches, then further reveals the potential of an RMS.

Reinforcement learning (RL) is a promising decision-making approach which had already led to breakthroughs in a lot of research aspects including game playing, robotics, finance, and autonomous driving. With a monolithic parametric reward function, RL agents addressed a wide range of complex tasks by integrating information from real manufacturing processes. Simulating complex and changeable production systems, like RMS, is an area where RL principles may be applied.

An end-to-end deep reinforcement learning framework was developed in this research. The resulting policy is trained to generate a sequence of consecutive actions that can be used as an RMS schedule to manage a fluctuating market simulator in real-time. Keywords:

Reconfigurable manufacturing system, flexible job-shop scheduling problem, system simulation, discrete-event simulation, reinforcement learning, deep Q network, proximal policy optimisation

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to all those who have contributed to the completion of my PhD thesis. Without their guidance, support, and encouragement, this accomplishment would not have been possible.

First and foremost, I must show my highest respect and honour to my supervisor Prof. Salonitis who always highlighted the most critical problems when I lose focus on my research. The constant guidance and unwavering support have been invaluable throughout this journey. I am deeply grateful for the time and patience Prof. Salonitis have invested in my education.

I would also like to thank my second supervisor Dr. Emmanouilidis whose valuable feedback and insightful suggestions have significantly enriched the quality of my papers and research. The guidance and encouragement have been instrumental in shaping my ideas and refining my arguments.

I am grateful to the external examiner Prof. Nikolaos Papakostas and internal examiner Dr. John Patsavellas, who have generously shared their expertise and provided invaluable feedback on my thesis. The comments and suggestions have been instrumental in shaping my research and strengthening my arguments.

I would like to express my appreciation to my friend Yijing Ji who helped me design the iconography for my thesis. The creativity and attention to detail have greatly enhanced the visual appeal of my work and smoothed the story line.

Although we didn't get to spend as much time together during my PhD journey, I am deeply grateful to my girlfriend Yali Wang, whose love and support have been a constant source of strength and inspiration. Even when we were apart, you always made time to listen to my complaint and provide words of encouragement. Your belief in me and your unwavering support have sustained me during the challenging times of this journey.

Finally, I would like to thank my parents whose unwavering support and encouragement have been a constant source of inspiration throughout my life. Your love and sacrifices have enabled me to pursue my dreams and achieve my goals.

Once again, thank you all for your invaluable contributions to my PhD journey. I am deeply grateful for your guidance, support, and encouragement.

TABLE OF CONTENTS

ABSTRACT	i
ACKNOWLEDGEMENTS.....	ii
LIST OF FIGURES.....	vi
LIST OF TABLES	viii
LIST OF EQUATIONS.....	ix
LIST OF ABBREVIATIONS	x
1 INTRODUCTION.....	11
1.1 Background.....	11
1.2 Reconfigurable manufacturing system.....	13
1.3 Challenges of RMS real-time scheduling	14
1.4 Research aim and objectives.....	16
1.5 Thesis layout.....	17
2 Literature Review	18
2.1 Scheduling for a reconfigurable manufacturing system	18
2.2 Deep reinforcement learning enabled schedule optimisation.....	28
2.2.1 Deep reinforcement learning milestones	32
2.2.2 Reinforcement learning in manufacturing system schedule optimisation	40
2.3 Key findings and research gaps.....	44
3 Training Framework	46
3.1 Introduction	46
3.2 Reconfigurable Manufacturing System Modelling.....	47
3.2.1 The fluctuating market simulator	47
3.2.2 The Core Components of an RMS	49
3.2.3 Modelling Approach.....	52
3.3 Reward Function for Multiple Objectives.....	55
3.3.1 Sparse Rewards and Dense Rewards	56
3.3.2 From Key Performance Indicators to Rewards.....	57
3.4 An Agent Can Schedule.....	58
3.4.1 An Off-policy approach.....	59
3.4.2 An On-policy approach.....	64
3.5 Other Advanced Tricks for Training	68
3.5.1 OpenAI gym Interface	68
3.5.2 Multi-environment & multi-processing	69
3.5.3 Hyperparameter Tuning and its Automation	69
3.5.4 Action Mask.....	70
4 Case Studies	71
4.1 Case One: Agent Validation on a dimensionless RMS	71
4.1.1 Experiment 1-1: Basic DQN Algorithm Validation	77
4.1.2 Experiment 1-2: Advanced DQN Algorithm Validation	79

4.1.3 Case Study One Summary.....	80
4.2 Case Two: A Grid-world RMS	81
4.2.1 Experiment 2-1: An Anti-work Agent	84
4.2.2 Experiment 2-2: Reward IS Critical	86
4.2.3 Case Study Two Summary.....	89
4.3 Case Three: Machines Know How to Pull Properly	91
4.3.1 Experiment with a PPO Agent.....	94
4.3.2 Case Study Three Summary	99
5 Discussion and Conclusion	101
5.1 Key take-outs, discussion, and generalisability.....	101
5.2 Key contributions to knowledge	104
5.3 Limitations and future research directions	106
5.4 Final remarks about potential industrial application	109
REFERENCES.....	110
6 Appendices.....	128
6.1 Appendix One: The source code GitHub Repositories.....	128

LIST OF FIGURES

Figure 1-1 A Reconfigurable Machine Tool and Modules.....	14
Figure 2-1 A camping scenario.....	30
Figure 2-2 The camping scenario in reinforcement learning context.....	31
Figure 2-3 Reinforcement learning in the most simplified form.....	32
Figure 3-1 General RL agent–environment Interaction	46
Figure 3-2 A stochastically updating order list with 3 products	48
Figure 3-3 A simplified reconfigurable manufacturing system	53
Figure 3-4 A Grid-world Shape RMS-DES	54
Figure 3-5 Deep Q Network Learning Process.....	60
Figure 3-6 Basic DQN for RMS-DES Scheduling	64
Figure 4-1 RMS02 ideal Schedule with Dual Product-0 Configuration	72
Figure 4-2 The RMS03 ideal schedule with 3 RMTs in Product-0 configuration	73
Figure 4-3 The RMS03 ideal schedule with 2 RMTs in Product-0 Configuration	73
Figure 4-4 The RMS03 ideal schedule with 2 RMTs in Product-1 configuration	74
Figure 4-5 The RMS03 ideal schedule with 3 RMTs in Product-1 configuration	74
Figure 4-6 Basic DQN Training Process Comparison (RMS01&02).....	77
Figure 4-7 Detailed Training Performance Improvement.....	78
Figure 4-8 Advanced DQN Algorithms Training Process	79
Figure 4-9 One AGV Movement Trajectory Sample	85
Figure 4-10 Experiment 2-2 Reward Trend	87
Figure 4-11 Productivity Trend	88
Figure 4-12 RMT-focused RMS Simulation Flowchart	93
Figure 4-13 Exploit Performance Reward Trend	95
Figure 4-14 Order Completion Record	96
Figure 4-15 Accumulated Inventory Record	97

Figure 4-16 Average Inventory Level for Individual Orders 97

Figure 4-17 Total Reconfiguration Times Record..... 98

LIST OF TABLES

Table 2-1 Pros. and Cons of Existing Optimisation Algorithms.....	29
Table 2-2-1 Algorithm 1: Q-learning Pseudo Code.....	34
Table 2-3 Algorithm 2: Deep Q-learning pseudo code	35
Table 3-1 AGV action space.....	50
Table 3-2 AGV-based MHS events	51
Table 3-3 Basic DQL Process of a Scheduling Agent	62
Table 3-4 Clipped PPO-based Scheduling Agent Training Scheme.....	67
Table 4-1 Case Study One Order List: Total 166 Product-0 and 362 Product-1	71
Table 4-2 Case Study One Reward Function	74
Table 4-3 States to Form an Observation	76
Table 4-4 Experiment 2-1 & 2-2 Settings	83
Table 4-5 Experiment Case 2-1 Reward Function.....	84
Table 4-6 Experiment Case 2-2 Reward Setting	87
Table 4-7 Case Study 3 Reward Function.....	93

LIST OF EQUATIONS

(2-1)..... 33

(2-2)..... 33

(2-3)..... 33

(2-4)..... 37

(2-5)..... 38

(2-6)..... 38

(3-1)..... 60

(3-2)..... 61

(3-3)..... 61

(3-4)..... 63

(3-5)..... 64

(3-6)..... 67

(3-7)..... 67

LIST OF ABBREVIATIONS

AMOS	Archived multi-objective simulated annealing
CNC	Computer numerical control
DES	Discrete-event simulation
DML	Dedicated manufacturing line
DP	Dynamic programming
DQN	Deep Q-Network
ERC-RMS	Engineering research centre for reconfigurable manufacturing systems
FIFO	First in first out
FJSP	Flexible job-shop scheduling problem
FMS	Flexible manufacturing system
GA	Genetic algorithm
ILP	Integer non-linear programming
LP	Linear programming
MA	Multi-agent
MDP	Markov decision process
MHS	Material handling system
MILP	Mixed-integer linear programming
MINLP	Mixed-integer non-linear programming
NSGA-II	Non-dominated sorting genetic algorithm II
PPO	Proximal policy optimisation
RL	Reinforcement Learning
RMS	Reconfigurable manufacturing system
RMT	reconfigurable machine tool
S	States
SA	Simulated annealing
TD	Temporal difference
TS	Tabu search

1 INTRODUCTION

1.1 Background

Faced with the need for throughput and cost efficiency, Henry Ford invented the moving assembly line in 1913, initiating the mass production paradigm. In the 1970s, Japanese manufacturing industry began to develop lean production principles. Since then, consistent quality of manufactured products has been one of the main attributes of manufacturing systems. In the late 1970s, with the development of Computer Numerical Control (CNC) machines, the Flexible Manufacturing System (FMS) paradigm was created, which made it possible to produce different products on one single manufacturing system.

In the early 1990s, researchers claimed that the world market was becoming unpredictable [1], with the proliferation of widescale globalisation, and that the dedicated manufacturing lines (DMLs) should be replaced by next generation machine lines as the competitive landscape changed. Manufacturing companies began to face unpredictable market changes, including rapidly changing product demand due to shorter lifecycles and frequent new product updates or launches.

At the time, FMS was, through expanding its boundaries, widely considered to be the next prominent manufacturing paradigm.

However, in order to increase the throughput through FMS, considerable investment is needed, and very little throughput growth is typically realised. This allows FMS to only cope with high value-added products with small batch size requirement [2].

These two attributes exposed a niche, which requires a new kind of manufacturing system. This niche requests a newer paradigm that can take advantages from both DMLs and FMSs. Such a paradigm made the design of manufacturing systems for new plants a major challenge, as it could affect plant performance for tens of years after the plant was designed. This paradigm also needs to be designed to respond quickly to unforeseen market surges and unpredictable product changes.

In response to this challenge, in 1995, Koren [3] proposed a new system architecture of factory design that he called “reconfigurable manufacturing system (RMS).”

1.2 Reconfigurable manufacturing system

Similar to recent manufacturing paradigms and architectures, such as the dynamic manufacturing networks [4–6], the RMS paradigm [7] was introduced to address the fluctuations and high uncertainty that characterise today's markets. The machines inside an RMS can also change its accessories, or namely modules, to make themselves reconfigurable machine tools (RMTs). The RMS paradigm enables a “live” factory whose structure changes cost-effectively in response to markets' needs, so it can keep supplying products at competitive price levels for many years after the factory design. Koren's proposal [7] to form an “engineering research centre for reconfigurable manufacturing systems” (ERC-RMS) at the University of Michigan was founded by National Science Foundation with \$30 million and industry with \$14 million in research on the impact of reconfiguration of manufacturing systems on system performance and on the theory and modelling of variable flow in product assembly [1]. The centre created the RMS theoretical foundation and invented RMS technologies that were implemented in the U.S. automotive and aerospace industries, enhancing thereby the industry competitiveness.

The study based on the RMS-ERC involved more than 30 companies. At the same time, researchers at the University of Iowa and the Massachusetts Institute of Technology have studied the design of such systems [1]. It is therefore foreseeable that a reconfigurable manufacturing system technology based on science will become a major technology to support the development of manufacturing in the early 21st century.

The developed technologies have been successfully implemented in the U.S. automotive companies —Ford, General Motor, and Chrysler — which have increased their system responsiveness [1] and created substantial economic value for these firms. RMS is not only an open-architecture manufacturing system that can respond to the challenges of globalisation [8], but also one that boosts productivity, enhancing thereby the competitiveness of manufacturing enterprises. RMS can also achieve agility and sustainable manufacturing [9].

1.3 Challenges of RMS real-time scheduling

In order to understand the challenges on RMS real-time scheduling, a simplistic example is provided. Consider an RMS that processes multiple types of parts by employing multiple autonomous RMTs. Each part contains diverse features and requires sequential operations. We assume that each line consists of several multi-stage RMTs which have different configures. As present in Figure 1-1 below, each machine can be reconfigured by swapping its modules.

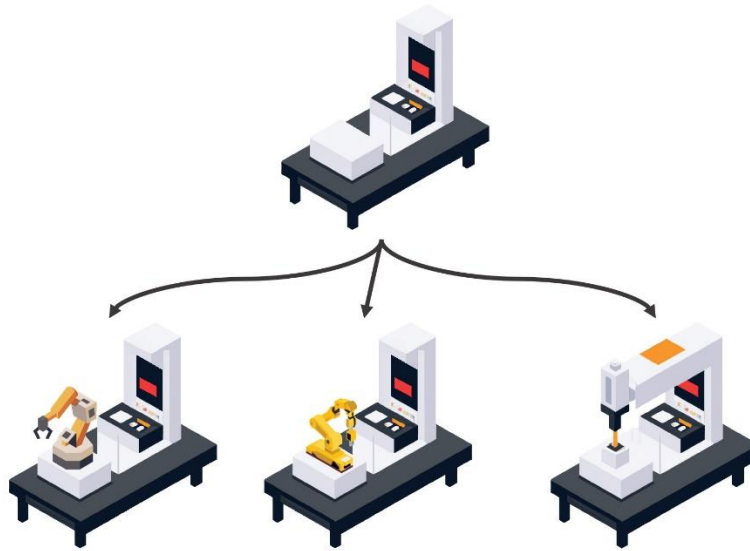


Figure 1-1 A Reconfigurable Machine Tool and Modules

We refer to the machines in different configurations as variants. To satisfy the demand, we determine the number and variants of the machines for each stage and assign to them the necessary operations. As the number of RMS configurations increases exponentially with the size of the line and considering the fact that demand changes frequently, two challenges emerge: how to efficiently select the optimal configuration with limited resources given, and how to allocate resources among lines. Because of the model complexity, finding near-optimal resource and demand allocation plans for these lines by a centralised management strategy is inefficient. In addition, the number of relocation decisions rapidly grows with increasing number of lines and types of resources. The resource and demand allocation challenges in RMS planning have become inadequate for known optimisation tools mainly due to the high

computational cost involved. Consequently, a novel and efficient method is imperative for addressing these complexities.

The reinforcement learning (RL) is drawing a lot of attention from both industry and academia recent years. RL provides a lot of potential on providing *super-human* performance in a decision-making problem with large observation space. This research applied RL to RMS scheduling.

1.4 Research aim and objectives

The overall aim of this research is to *develop a reinforcement learning-based decision-making policy training framework for reconfigurable manufacturing system scheduling*.

In order to achieve the research aim in a systematic and structured manner, the aim has been separated into individual objectives. These objectives progressively drive the research closer to the overall goal. The individual research objectives are:

RO-1: Develop a thorough understanding of RMS scheduling through a critical analysis of the relevant literature. This step will lead to the identification of the key challenges of the RMS scheduling problem and how RL algorithms are applied to solve similar scheduling issues.

RO-2: Develop a framework to identify interfaces amongst simulation models, reward functions, and RL-agent.

RO-3: Develop RMS simulation models to create a comparison baseline with conventional rule-based scheduling. Such models with corresponding reward functions form the RL training environments for developing new RL-based scheduling policies.

RO-4: Deploy RL algorithm candidates to identify the most appropriate approach to embrace the characteristics of the RMS scheduling problems.

RO-5: Validate the framework through case studies.

1.5 Thesis layout

The rest of thesis is organised as follows.

In the next chapter, a comprehensive, critical review of the state-of-the-art on contemporary scheduling solutions for RMSs is presented, and the mechanism is outlined about how RL may support decisions in reconfigurable manufacturing systems. Due to relatively few papers that consider the application of reinforcement learning in the context of RMS, other types of changeable manufacturing systems are also included. In Chapter 3, the end-to-end framework is proposed and discussed in detail. The framework starts with designing an order generator to simulate the fluctuating market. Then two different approaches employing discrete-event simulation to model an RMS are presented and discussed. How to transfer the objectives into a reward function is then highlighted. Two different approaches to train an RL agent is explained at last.

Chapter 4 corresponds to the framework validation. In this chapter, the validity of the framework is established through three case studies. Then, the validity of the framework is enhanced through several sensitivity analysis experiments, which are carried out to explore the impact of changing the values of certain key parameters on the performance of the system. These variables covered four major categories include system parameter, reward function, agent structure, and advanced agent training techniques.

Chapter 5 summarises the preceding chapters, then highlights the key findings from all three case studies. The limitations of this research are discussed, and then future research directions are provided.

2 Literature Review

This chapter analyses the contemporary literature in light of the overall research objectives. The current chapter is organised as follows: the concept and features of a reconfigurable manufacturing system (RMS) are discussed first, while its history is briefly reviewed. Then the review focused on two branches, the first one is the scheduling of an RMS, and the second one is reinforcement learning (RL) aided manufacturing system schedule optimisation.

The first section looks at how other researchers consider the RMS scheduling problem as an extended flexible job-shop scheduling problem (FJSP) using exact algorithms and heuristics or meta-heuristics to optimise the RMS schedule.

2.1 Scheduling for a reconfigurable manufacturing system

Since Henry Ford realised the mass production paradigm through the assembly line in 1913, the manufacturing paradigm has evolved over the last 100 years to accommodate the characteristics of modern economic and social environments. As one of the most critical concerns of manufacturing and production management, production scheduling has been researched and studied extensively in the literature. Including allocating and sequencing available resources to tasks, an optimal product schedule meets all constraints while optimising specific objectives [10]. More than half a century ago, researchers found that the job-shop scheduling problem (JSP) considering setup times is NP-hard [11]. On top of JSP, a Flexible JSP (FJSP) considers a set of available machines instead of for one or more tasks. As FJSP increases the flexibility of a manufacturing system, the complexity of the scheduling process increases. As the manufacturing paradigm becomes more complex, the scheduling techniques start falling short of the systems' requirements and objectives.. As Tang et al. summarised [12], RMS core characteristics bring more capabilities while a smart and novel way to schedule is needed.

RMS Literature is ample and covers many areas like configuration design, control process planning, layout optimisation, and production scheduling. All six

key characteristics of RMS boost its functionality to react to demand rapidly. Therefore, effectively arranging human and physical resources becomes complex in such a dynamic environment. Scheduling and process planning can usually be formulated into dissociated and integrated functions in RMS. However, unlike process planning, scheduling alone in the RMS context is not a very active topic, yet some trials on scheduling in RMS have been published throughout these last two decades. As scheduling and process planning share some common problem-formulating paths, this section will also keep an eye on supporting process planning in RMS.

The general pattern for researchers to solve an RMS scheduling or process planning problem has two steps. The initial step is usually formulating the problem into a mathematical model. Then, after transforming and abstracting the problem, the following step is seeking the exact solution through mathematical programming or finding an approximated result considering the complexity of the system.

Starting with simpler the easiest linear programming (LP) models [13–15], researchers keep upgrading their models to integer non-linear (INLP) [16,17], mixed-integer linear (MILP) [18,19], and more advanced mixed-integer non-linear (MINLP) models [20,21] to describe how an RMS operates.

Dynamic programming (DP) was used in some scheduling problem instances.

DP and RL are closely related and share many similarities. While DP is a mathematical optimisation technique used to solve problems that can be broken down into smaller sub-problems, where the optimal solution to the overall problem can be found by combining the optimal solutions to the sub-problems.

RL, as a subfield of machine learning, focuses more on developing algorithms that can learn to make decisions based on feedback from the environment. It is often used in situations where the optimal decision-making strategy is not known, and the agent must learn by trial and error.

Both DP and RL involve finding optimal solutions to problems by breaking them down into smaller sub-problems and combining the solutions to those sub-problems. However, there are some key differences between the two:

1. DP assumes that the entire problem is known, including the underlying model and the rewards associated with each state and action, while RL does not require knowledge of the underlying model.
2. DP is typically used for problems with a small number of states and actions, while RL is better suited for problems with a large number of states and actions.
3. DP assumes that the environment is stationary, while RL can handle non-stationary environments where the rewards and transition probabilities can change over time.

Despite these differences, DP and RL are both important tools in the field of artificial intelligence and are often used in conjunction with each other. RL algorithms can be seen as a way to solve problems when the underlying model is unknown, while DP can be used as a benchmark to evaluate the performance of RL algorithms when the underlying model is known. This makes RL more future-oriented as it can handle complex, non-linear problems. DP is well-suited for problems with a small number of states and actions, but as the size of the problem increases, the computational complexity of DP can become prohibitively high. RL, on the other hand, is able to handle much larger and more complex problems by using techniques such as function approximation and deep neural networks. RL can learn without an explicit model while DP requires an explicit model of the environment, including the transition probabilities and rewards associated with each state and action. In RMS applications, however, this model may be difficult to be translated into a pure mathematical form. RL can learn directly from experience by interacting with the environment, without the need for an explicit model. RL can adapt to changing environments while DP assumes that the environment is stationary, meaning that the transition probabilities and rewards associated with each state and action do not change over time. RMS was proposed to deal a fluctuating market, dynamic and stochastic environment in another way, which makes DP

has few potential. RL has another advantage which is RL can learn online. DP requires the entire problem to be known in advance, including the rewards and transition probabilities associated with each state and action. In RMS and many other real-world applications, however, the rewards and transition probabilities may not be known in advance or may change over time. RL can learn online, meaning that it can continually update its policy based on new experiences, without the need for a complete model of the environment.

Heuristic and meta-heuristic approaches are commonly used as optimal solution approximators for more complex scheduling problems, not only in RMS. These algorithms usually explore the solution space efficiently with advanced and diverse strategies. Tabu search (TS) [22,23], simulated annealing (SA) [24–26], and genetic algorithm (GA) [8,27–33] are the three popular approximators among RMS scheduling and process planning researchers. TS is a local search-based meta-heuristic that discourages revisiting previous solutions by incorporating an adaptive memory. Another meta-heuristic algorithm SA is inspired by metallurgical annealing techniques. Metallurgists suggest that defects and residual stress can be minimised if a material gets heated and then cools down slowly. SA algorithm slowly decreases the probability of worse solutions while the solution space is under exploration. Archived multi-objective simulated annealing (AMOS) [26] is designed for multi-criteria optimisation. GA is a popular evolutionary algorithm inspired by natural screening and selection. The potential encoded solutions in the initial GA-based approaches, called individuals, usually represent its genotype as a string of 0 and 1. Most modern approaches use more complex representation, but the principle keeps the same. The group of these individuals is called a population. A population generates offsprings and new individuals through biological-like operations, such as crossover and mutation. The offsprings corresponding to better performance may be part of a new population in the next iteration. The continuous evolution will continue until the desired quality is reached or iterated into a stop condition. The Non-dominated sorting genetic algorithm II (NSGA-II) [34–36] is one of the most popular.

Li and Xie [32] proposed a genetic algorithm embedded with extended timed-place Petri nets so that the modules can completely model all possible behaviours of a reconfigurable production line. Furthermore, they oriented the crossover and mutation operators towards the elements of the Petri nets model instead of the ones in the problem space.

A GA-based integrated scheduling method proposed by Ye and Liang [31] allows simultaneous process scheduling for modular products while selecting the optimal RMS configurations. First, the method presents a schedule in a genotype form. Then, the manufacturing cells are displayed as sub-genotypes with configuration and job schedules for the corresponding product variant.

Spicer and Carlo [37] tackled the scalability of an RMS in order to minimise the investment and reconfiguration cost through a combined integer programming/dynamic programming approach. This approach optimises the configuration path for further scheduling.

To minimise the life cycle costs of automated reconfigurable assembly lines, Bryan et al. [28] proposed a GA-based scheduler for a staged network of reconfigurable assembly system planning and validated it on pure parallel, pure serial, and mixed setup. The authors generalised this approach in [38] for decreasing product development time. The comparison between DP and GA suggests that DP only performs better in small-scale applications.

Galan [39] scheduled 'product families' on RMS. Since RMSs are generally dedicated to product families, the authors propose an approach to group products into families and then schedule these families to minimise the total cost. Metaheuristics are used to solve the problem.

Bryan et al. [40] employed a DP algorithm to minimise cost, and the result is significant worse than the GA-based algorithm.

Abbasi and Houshmand [41] used a TS-based algorithm aimed at profit maximisation to determine an optimum sequence of production tasks (represented by a genotype with two sets of interrelated genes) corresponding to configurations and batch sizes. The authors further investigated [8]

maximising profit rate through a GA-based optimisation of production task arrangement considering multiple objectives, including optimum product family configurations, span, and production sequence.

After getting a potential optimal production task sequence by a meta-heuristic, genetic algorithm (GA), approach, Abbasi and Houshmand [42] applied the MINLP model to get the final optimal result for maximising general profit.

Targeting the lowest manufacturing cost and completion time, Chaube et al. [34] proposed an RMS planning and scheduling multi-objective NSGA-II approach for assigning multiple part variants to available machines.

Considering the scalability of an RMS, Carlo et al. [43] developed the ordering policy machine configurations while simultaneously minimising investment and operating costs using a GA-based approach.

Based on simulated annealing (SA) with auxiliary knowledge, Musharavati and Hamouda [25] put forward a multiple-part process RMS scheduler to optimise the demand respected throughput ratio while minimising operating cost.

Musharavati and Hamouda [24] proposed a simulated annealing (SA) based algorithm coupled with auxiliary knowledge exploitation and parallel architecture to generate enhanced process plans in RMS. The computational speed and effectiveness of the algorithm are satisfactory.

Nehzati et al. [44] presented a fuzzy-based scheduling model. The authors deal with the job assignment problem in a logical (soft) RMS. The model selects the best alternative machine with a multi-criteria method. The proposed framework clearly presents the subjective input variables and its job priority output. In order to calculate minimum investment cost, Carlo et al. [43] abstracted multistage scalable RMS into an ILP model. A GA-based heuristic is presented to solve such ILP then retrieved optimal solution in 24 test instances.

Eguia et al. [23] used Tabu-search after formulating a scheduling problem of a reconfigurable cellular manufacturing system into a MILP model.

Borisovsky et al. [27,45] considered a reconfigurable production line that had a sequence of CNC machines. Focused on minimum total equipment cost, a GA-based schedule algorithm for a machining line balancing problem with industrial constraints including total workstation number, workstation machine number, and machine assignment number. The algorithm uses a permutation representation and a heuristic decoder, along with a problem-oriented mutation operator. A MIP decoder is used after the GA scheduler gives a permutation to construct the optimal local solution.

In order to construct a schedule for a practical RMS in a dynamic market, Li et al. [30] proposed a GA-based reconfiguring scheduling algorithm while considering the part-releasing sequence.

Similar to the previous researchers, Borisovsky et al. [45] presented a MILP model to tackle a reconfigurable machining line balancing problem before further GA optimisation. In this paper, the authors used a DP algorithm to minimise the total setup time with optimal operation sequence.

Using the archived multi-objective simulated annealing (AMOS) approach, Bensmaine et al. [46] addressed an RMS process plans generation problem from a multi-objective perspective. The experimental results, obtained from Pareto fronts, are compared and classified into a preferential rank based on the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS). Based on the process plans, these researchers later [33] considered the problem of RMS's design based on product specifications and reconfigurable machines capabilities. The problem is related to selecting reconfigurable candidate machines among an available set, which will then be used to optimise total completion and cost. The selection of the machines considers two main objectives, the minimisation of the total cost (production cost, reconfiguration cost, tool changing cost and tool using cost) and the total completion time. An adapted version of NSGA-II is proposed to solve the problem. A specific heuristic is proposed and studied by these researchers later [19] for an integrated process planning and scheduling problem considering RMS's multi-configuration nature. The discrete-event simulation model describes an RMS

with multiple RMTs. The goal is to tune jobs assignment based on processing time and RMT availability. The optimal multi-unit macro-level process plans target minimum total completion time. The experimental results suggested that this specific meta-heuristic approximating method can find high-quality solutions rapidly, even if it is based on a relatively simple search rule.

Yu et al. [47] considered a practical constraint: the numbers of pallets/fixtures in the system are limited, so a part can be released into the system only when a required fixture is available. They use a priority rule-based approach to solve the problem.

Choi and Xirouchakis [48] proposed an RMS holistic production planning mathematical model and optimised it by employing multi-objective LP. The planning approach's objectives are to maximise throughput while minimising energy usage and environmental impact.

Rabbani et al. [49] presented a bi-objective MINLP schedule model for maximising the machine utility rate and minimising the total reconfiguration and internal movement cost. The authors utilised a goal programming technique to tackle the multi-objective optimisation.

Reconfigurable manufacturing tool (RMT) is an important component inside an RMS. When requirements change, an RMT can implement cost-effective reconfiguration to realise most of the key characteristics of an RMS. With a given demand, Moghaddam et al. [50] propose a MILP model to help select optimal RMT configurations. This reconfiguration cost-minimising approach has been extended by Moghaddam et al. [51] with two new, improved MILP approaches and one ILP approach to help decrease design cost, reconfiguration number, and idle capacity.

Combining NSGA-II, the Technique for Order Preference by Similarity to an Ideal Solution (TOPSIS), and Shannon entropy weight, Goyal et al. [36] proposed an RMT configuration optimising approach based on NSGA-II. The approach aims at maximising machine capability while minimising the reconfiguration cost. This approach has been extended by Ashraf and Hasan

[52] by additionally considering reliability. The approach helps to tune the RMS in a single-part flow setting and performs well under multiple demand scenarios.

After proposing a 4-product automobile reconfigurable production line layout, Prasad and Jayswal [53] used an integrated approach of the Reference Ideal Method and Shannon entropy to optimise due date, profit over cost, and reconfiguration effort.

To increase flexibility and scalability, Ghanei and AlGeddawy [54] considered a scheduling problem as a MILP model and optimised the schedule based on energy price variation.

Dou et al. [35] designed a GA-based scheduling method that targets capital minimisation, reconfiguration cost, and total tardiness of an RMS. The method can improve the schedule of multiple products and corresponding configurations. These researchers improved this approach [29] and applied it to two industrial cases running on a one-part multiple-process flowline setup. This method is validated through comparison against ant colony optimisation, particle swarm optimisation and an edge selection-based GA approach. In [55], this research team outperformed the previous GA-based method as well as the aforementioned GA approaches [20,21]. The new multi-objective particle swarm optimisation-based approach delivered the best cost-effective result based on the newest MILP model.

Benderbal et al. [56] tackled a machine-selecting problem by proposing an NSGA-II-based algorithm to maximise the system flexibility, while minimising the completion time. The authors then [26] used the archived multi-objective SA method to measure and optimise the modularity of the RMS. These two approaches may improve balance modularity, operation time, machine utility, and cost.

Asghar et al. [57] proposed a generic co-evolution scheduler for process planning and configurations. The weighted GA optimising approach focuses on market demand fluctuation and reconfigures the RMS to the most capable form for manufacturing a part family.

To tackle a configuration design and schedule problem of a multi-part flowline RMS, Dou et al. [21] developed a MILP model, which aims at minimising earliness and tardiness penalty, cost variation, and total operation cost. This model is then optimised [20] and tested to get Pareto optimal solution through the ϵ -constraint and GA-based methods. The ϵ -constraint did not perform well in large-scale trials.

While designing a 4-configuration RMT, Mahmoodjanloo et al. [58] formulated a two-machine RMS scheduling problem into two MILP models. One is based on operation position, and the other is based on operation sequence. Considering a plug-and-play reconfiguring process, researchers demonstrated the superiority of the sequence-based model over the position-based model. Researchers did extensive experiments on a large-scale setup with unlimited modules for RMTs. The optimum schedule was found using a proposed Nelder-Mead mutation strategy employing a self-adaptive differential evolution algorithm. Considering a dynamic environment, these researchers [59] extended the RMT to a distributed system. The scheduling problem of such a system became very complex, and they addressed its rescheduling problem by using a meta-heuristic method called equilibrium optimiser (EO). Bortolini et al. [60] also extended and generalised the scheduling problem in [61] to a multi-product multi-period setup by integrating auxiliary module selecting and assigning problems. Though the role of module assignment in production scheduling was not discussed, researchers built a model of a cellular RMS with limited modules. The proposed model then gets optimised targets on minimising part flow time, module flow time, reconfiguration time, and balancing available modules throughout time. To mitigate the COVID-19 impact on human recourse, Vahedi-Nouri [62] remodel the previous approaches thoroughly by considering infectious disease health concerns and restrictions. The proposed MILP and constraint programming model additional parameters like workers' vulnerability, health safety risk with machines and other epidemiological restrictions.

The literature above shows that process planning and scheduling in RMS are usually conflicted. However, both are core activities that respond to realising any product from raw material to finished goods. Scheduling is typically bounded by restrictions on process planning while balancing due dates, costs, and, most importantly, resource availability, which process planning typically ignores. As one of RMS's most basic components, RMT provides the most convenient approach to enrich RMS core characteristics. Despite being specifically designed for machining, assembly, inspection, or fixturing, RMTs poses a challenge to the current scheduling strategies and optimisation methods. These platforms with auxiliary modules enable operational flexibility or increase processing speed while their assembly and disassembly processes could be time-consuming and delay the production progress. All these features make RMS an important tool for dealing with complex process plans. Although some integrated approaches consider different objectives among process planning and scheduling simultaneously, the algorithms proposed are only efficient in very limited cases.

2.2 Deep reinforcement learning enabled schedule optimisation

Manufacturing companies nowadays, in the age of Industry 4.0 [63], are more than ever being challenged towards transforming their activities into smart and connected operations.

For example, manufacturing systems gradually integrating smart technologies may achieve better performance while allowing the more efficient monitoring of their status.. These (smart) systems have better real-time data capturing and acquiring capabilities to reveal more about the status of components inside a system.

With all these capabilities, production scheduling of a manufacturing system is playing a more prominent interdisciplinary role across industrial engineering, automation, and management science, amongst other fields. In the previous section, the two conventional optimisation algorithms, exact algorithms, heuristics and meta-heuristics, are briefly discussed. The pros and cons [64] of these two main method categories are summarised in Table 2-1 below.

Table 2-1 Pros. and Cons of Existing Optimisation Algorithms

Methods / Algorithms	
Exact Algorithm	Heuristics & Meta-heuristics
Pros.	
Optimal solution guarantee	Less computationally expensive
	Problem-independent and can be applied to a wide array of optimisation problems
	Provides acceptable solutions within a reasonable amount of time
Cons.	
Difficult to solve large-scale NP-hard problems	Optimality is not guaranteed
	Proximity to the optimal solution cannot be determined
Over-simplifications are required and might impact the reliability of the solution	Prone to be trapped in local optima
	Deep understanding needed

The prospering artificial intelligence (AI) and machine learning (ML) disciplines bring new hope for operators to handle more complex optimisation problems characterised by unpredictable events, then enable them to pursue agile responses.

Reinforcement learning (RL) is under the three main ML categories, alongside supervised and unsupervised learning [65]. The recent field of deep reinforcement learning (DRL) has made significant achievements in sequential decision-making problems within non-linear complex environments with large observation spaces like games [66–69], robot control [70,71], short-term weather predictions [72], nuclear science [73], mathematics [74], and, manufacturing scheduling problems which will be reviewed below. Henceforth in this paper, deep reinforcement learning will be the default mode unless explicitly specified otherwise.

The core concept of RL is simple: enlighten an agent by interactions. To explain this, a simplistic scenario might help convey the core principles of RL:

Suppose you drive to a campsite for an overnight camping trip with your family. You find yourself in the middle of a small piece of a clearing. You hear the

sound of a creek coming from behind the bushes, some tweeting birds fly pass overhead, your stomach starts generating growling sounds, your partner is opening the back trunk while children are playing around the trees. A visualised scenario presents in Figure 2-1 below.



Figure 2-1 A camping scenario

Now, imagine how many activities are at hand: you could start building the tent, complain about the long-drive journey, light up some charcoal for the grill, reproach the covertly crawling children from a squirrel while deciding whether to take out an energy bar from your backpack.

At such a moment, so many long-term or short-term concerns hover in your mind: re-energising your body, a happy holiday, a harmonic partnership, and happy children, amongst others.

The previous scenario fully covers all RL key components. You, the **agent**, gather all **observations** from a dynamic **environment**, then take action instantly according to your experience toward the most accumulated **reward**.

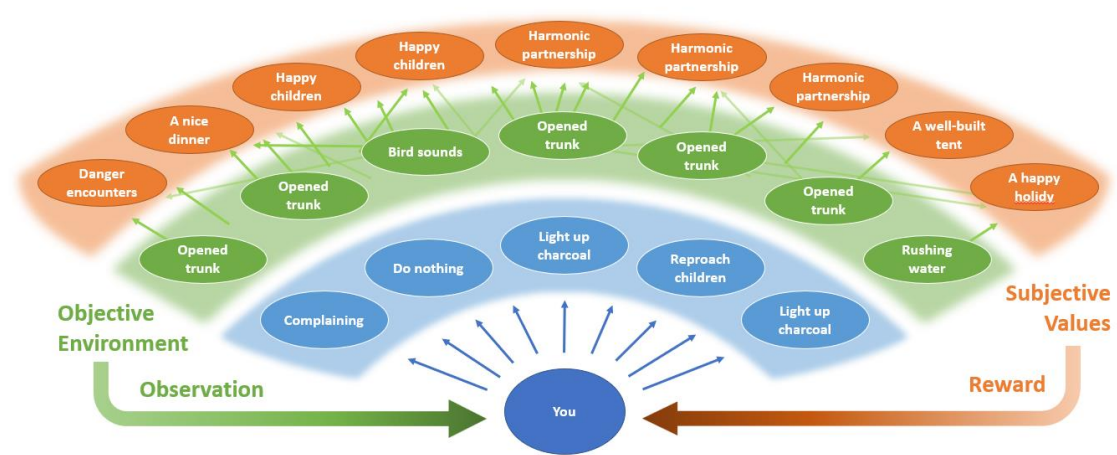


Figure 2-2 The camping scenario in reinforcement learning context

Now that the basic idea of an RL problem has been explained with the help of the previous scenario, it is time to present and uncover the founding governing concepts and their corresponding formal terminologies with reference to the example depicted in Figure 2-2.

First, an agent with policy π , “you” in the scenario, is the main body where the training/learning process happens. Policy π stores all the agent’s knowledge and describes, in most cases not explicitly, how this agent will react to the second basic term in RL. The second key element is the state s which is the input of an agent. A state at a specific time s_t describes how much information an agent can get in order to make a decision. The explicit decision shows as an action which is the third basic term in RL. An action a_t is the output of an agent in reaction to s_t . All possible actions of an agent can be represented in the form of an action space $\mathcal{A}$ which describes how an agent interacts with the environment. As the measure of interaction, the last basic RL component is introduced as reward $r(s, a)$ which is a feedback signal from the environment by taking action a at state s . A reward is generated through a reward function when a positive value that presents a training goal is fulfilled, while a non-positive one represents otherwise.



Figure 2-3 Reinforcement learning in the most simplified form

In summary, as depicted in Figure 2-3, RL mimics how humans or animals learn by defining an *agent*. This computational system performs autonomous actions to interact with a dynamic environment regarding a specific *observation*. The enlightenment of such an agent is learning to select optimal or sub-optimal actions to accumulate and maximise long-term *reward* through trial-and-error interactions with a dynamic environment. When an RL problem uses a deep neural network to approximate the value of any of the aforementioned components, even including the environment model, the entire method can be called deep reinforcement learning (DRL). The history of DRL will be briefly reviewed below, with some milestones that have attracted worldwide attention. Then some cutting-edge RL applications on manufacturing system scheduling will be reviewed and discussed.

2.2.1 Deep reinforcement learning milestones

The central and novel idea of modern reinforcement learning (RL) is derived from temporal-difference (TD) learning [75]. As the algorithm with the lowest space complexity and time complexity, TD learning combines the random sampling approach idea from Monte Carlo (MC) method, which estimates the state transition probability by massive but not exhaustive sampling [76], and bootstrap idea from dynamic programming (DP), which updates its estimation based on other estimation without waiting for the final outcome [77].

In an every-visit MC method [78], a new estimation of a non-terminal state $V(S_t)$ of a value function v_π for a given policy π can be written as follows,

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \quad (2-1)$$

Where α is the step-size constant that represents the time interval. α helps estimate an actual return G_t following the time t . TD-learning avoids waiting time t by replacing G_t with an observed reward from the next time step R_{t+1} and the $V(S_{t+1})$ so that the update can be written as

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (2-2)$$

Where γ is a discount factor, which is a positive number small or equal to 1, to weaken and balance the influence of a reward from the distant future.

One of the first milestones of TD learning is Q-learning. Watkins et al. [79] fully integrated DP's state transition equation, Bellman's equation, with online learning by formulating a discrete, finite world into a Markov decision process (MDP) problem. In Q-learning, the agent's policy π stores knowledge in a table. This table contains an exhaustive list of states and possible actions. To select the best action from the action space, Q-learning measures the expected reward (i.e., quality) of action, called action-value, typically referred to as $Q(S, A)$. The table that stores this knowledge is known as Q-table. The quality of best action at a certain state is referred as $Q^*(S, A)$. To obtain true Q^* , Q-learning updates its Q value using the following formula:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (2-3)$$

The epsilon-greedy method, which balances exploration and exploitation by choosing the highest expected reward action with probability $1-\epsilon$ and a random action with probability ϵ [75], helps Q-learning have a minimal probability to explore unknown actions. The basic Q-learning process can be written as the pseudo-code below:

Table 2-2-1 Algorithm 1: Q-learning Pseudo Code

Hyperparameters Input:

step-size / learning rate $\alpha \in (0,1]$, small $\varepsilon > 0$, episode limit M

Output:

*An optimal Q-table Q^**

```

01   Initialise a Q-table  $Q(s, a)$  arbitrarily except  $Q(\text{terminal}, \cdot) = 0$ 
02   for episode = 1 to  $M$  do
03       Initialise an initial state  $S_0$ 
04       for  $t = 0$  to terminal do
05           Choose an action  $A_t$  according to  $S_t$  by  $Q^*(s, a)$  or a random action by  $\varepsilon$ -greedy
06           Observe reward  $R$  and  $S_{t+1}$  by taking  $A$ 
07            $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$ 
08            $S_t \leftarrow S_{t+1}$ 
09       end for
10   end for

```

An intrinsic feature of the Q-table, which contains all state-action pairs, is that it raises the “curse of dimensionality” when introducing complex planning, decision-making, and advanced control problems. In what is historically considered as the first book dedicated to explaining practical applications of neural networks (NN) and dynamic programming, Dimitri Bertsekas and John Tsitsiklis [80] created the term “neuro-dynamic programming” to refer to combining neural networks as estimators and dynamic programming. The methodology also introduced reinforcing agent behaviour performance through iterative simulation, formed in Markov Decision Process (MDP).

By employing NN as a non-linear estimator of a Q-table, Minh et al. [81] from DeepMind expanded TD-learning. They developed deep Q-learning to help machine-learning start approaching the human-player level. Deep Q Network (DQN) enables an experience replay memory to increase data efficiency while breaking the strong correlations from consecutive samples, which may lead to inefficient updates.

The pseudo-code of the DQN algorithm shows in Table 2-3 below

Table 2-3 Algorithm 2: Deep Q-learning pseudo code

Hyperparameters Input:

*Discount Factor γ , step-size / learning rate $\alpha \in (0,1]$, Episodes Limit M ,
Batch Sample Length L , small $\varepsilon > 0$, capacity of replay memory N ,*

Output:

*An accurate non-linear quality function estimator Q^**

```

01   Initialise a state-action-value function  $Q(s, a)$  with random weights  $\theta$ 
      Initialise a replay memory  $\mathcal{D}$  with capacity  $N$ 
02   for episode = 1 to  $M$  do
03       Initialise / Observe an initial state  $S_0$ 
04       for  $t = 0$  to terminal do
05           Choose an action  $A_t$  according to  $S_t$  by  $Q^*(s, a; \theta)$  or a random action by
               $\varepsilon$ -greedy
06           Observe reward  $R_t$  and next state  $S_{t+1}$  by taking  $A_t$ 
07           Store transition  $(S_t, A_t, R_t, S_{t+1})$  into  $\mathcal{D}$ 
08           Randomly pick  $L$  samples  $(S_j, A_j, R_j, S_{j+1})$  from  $\mathcal{D}$ 
09           Set a training target  $y_j$ 
              
$$y_j = \begin{cases} R_j & \text{for terminal } S_{j+1} \\ R_j + \gamma \max_{a'} Q(S_{j+1}, a'; \theta) & \text{for non\_terminal } S_{j+1} \end{cases}$$

10           Perform a gradient descent step on  $(y_j - Q(S_j, A_j; \theta))^2$ 
11       end for
12   end for

```

The DeepMind researchers constructed a DQN controlling agent for Atari-2600 games with convolution to fully connect the NN structure in 2013 and validated the DQL algorithm. After involving the RMSProp algorithm and line ε -annealing, Minh et al. [82] employed deep reinforcement learning (DRL) and were able to demonstrate human-level controlling competence in Atari 2600 games.

DQN inherits a downside of Q-learning: overestimating the Q-value easily in a large state space setup. This overestimation is brought by Q^* . Unfortunately, it is hard for DQN to evade this inherent attribute, even if the neural networks were flexible and brought very little progressive error because the involvement of neural networks is targeted at an agent's need to deal with large state-action pairs. Targeting on eliminating the bias to choose overestimated actions

brought by *max* operation, van Hasselt [83] decoupled action-choosing and action-evaluating by separating one Q-network into two Q functions Q^A and Q^B . The proposed method is called double Q-learning. The action-choosing of double Q-learning is the same as Algorithm 2 – 05 based on Q^A . However, instead of evaluating the action by Q^A as Q-learning in Algorithm 2 – 09 would do, double Q-learning estimates the action by the target value function Q^B which learn from separate sets of experiences. Hasselt et al. [84] expanded double Q-learning to double deep Q networks (DDQN) the year after DQN was invented.

Prioritised Experience Replay (PER) [85] and duelling neural networks [86] are two advanced DQN techniques developed back-to-back. PER upgrades the way of storing and sampling experience in Algorithms 2 – 07 & 08. DQN stores experience sequentially and sample them uniformly. However, some experiences with greater TD-error may be more important than others. PER stores experience into a ‘sum-tree’ data structure which can rank experiences based on their TD-error. PER samples experiences with higher priority and significantly improved training efficiency. A dueling neural network is another simple yet effective modification for DQN. It focuses on improving the state-action function Q itself in Algorithm 2 – 01. The original Q network outputs the quality estimation of every action according to the input state. Dueling neural networks separate the state-action quality estimation into two parts, state value and action value. The sum of the state value and action value is the Q value. Updating the state and action networks separately can reduce the computational load because it does not require sampling all state-action pairs. Once a state value function gets updated, all state-action pairs get updated simultaneously.

The world-renowned AlphaGo, developed by DeepMind [69], defeated the best human player in the world in the board game Go. This became the most controversial intelligence event after Deeper Blue defeated Kasparov in international chess back in 1997. AlphaGo did prior work, using supervised learning, on predicting expert moves and then training the RL agent with actor-

critic architecture [87], which contains a policy-network and value-network. Considering Go's 19×19 grid and other human experience data, AlphaGo uses a [19×19×48] image-like to represent the policy network input and outputs the probability of every 362 moves. The move with the highest probability is the move that AlphaGo will take. The following year, the successor of AlphaGo, AlphaGo Zero, merged two individual policies and value neural networks in AlphaGo into one [68]. AlphaGo Zero also skipped the imitation learning part, reduced the state input to [19×19×17], and removed all human-designed features. These simplifications did not weaken the agent; on the contrary, AlphaGo Zero greatly improved and surpassed AlphaGo's performance with only 3-day training. Soon after that, Silver et al. [88] generalised the same algorithm into a single AlphaZero algorithm and then applied it to two strategy board games: chess and shogi. AlphaZero achieved superhuman performance within 24 hours with self-play. The newest member of the AlphaGo family, MuZero, developed by Schrittwieser et al. [89], transformed the input states into hidden states for training a representation network h_θ . By performing Monte Carlo Tree Search, as other members do on games' rule trees, on h_θ , MuZero is highly malleable to apply in a wide range of environments and achieved superhuman levels in many games.

The actor-critic architecture used in the AlphaGo family also inspired policy-based RL methods. Different from the value-based RL methods, like Q-learning, policy-based RL methods skip learning the values of actions and then choose an action based on its action value Q . Q may not be avoided while training a parameterised policy π . Policy-based RL methods do not have to use Q to select actions. The output of a policy function $\pi(a|s)$, a probability density function, is the probability of all actions. The state-value function can now be written as follow

$$V_\pi(s) = \sum_a \pi(a|s) \cdot Q_\pi(s, a) \quad (2-4)$$

As an action picker (actor), a policy network, $\pi(a|s; \theta)$, uses a neural network to approximate $\pi(a|s)$ where θ represents the trainable parameters inside a net. The value network (critic) $q(s, a; w)$ approximate the $Q_\pi(s, a)$ where w are the trainable parameters. Again, the value-network (critic) only helps train the policy-network (actor); it does not help in choosing an action.

$$V_\pi(s; \theta, w) \approx \sum_a \pi(a|s; \theta) \cdot q(s, a; w) \quad (2-5)$$

Hence at any moment

$$V_\pi(s_t; \theta, w) \approx \sum_a \pi(a|s_t; \theta) \cdot q(s, a; w) \quad (2-6)$$

The sum of the probabilities of all actions should be equal to 1 as the π is a probability density function which can be written as $\sum_{a \in \mathcal{A}} \pi(a|s; \theta) = 1$ where $\mathcal{A}$ is the action space. The training purpose is to maximising V_π .

Asynchronous Advantage Actor-Critic (A3C) and its synchronous version Advantage Actor-Critic (A2C) are trending policy-based RL algorithms developed by prominent industry practitioners. Mnih et al. [90] involve multiple workers in parallelised actor-critic training asynchronously. This parallelised training paradigm, A3C, targets improving training efficiency with previous reinforcement learning algorithms. Recheck the MDP of RL. An agent forms a state transition by observing the current state of the environment, interacting with the current environment through an action, getting a reward and, in addition to the reward, the observation of the next state is realised. This is a serial process. Suppose the agent tries to make full use of historical experience in order to reduce the bias caused by environmental perturbations, the variance of the model. In that case, an experience replay must be maintained just as DQN. However, this pool of experience would take up a huge amount of storage space; hence the algorithm can only be an off-policy approach. Off-policy approaches like DQNs sometimes take dozens of hours to train a simple agent for Atari-2600. With the additional parameters tuning, a good model could cost

too few experiments in a very long time duration. This happened to the author of this thesis, as will be explained and discussed in subsequent sections.

On-policy approach uses only the transactions generated by the current policy. Engaging the sampling batch often used in other neural network training techniques to stabilise the networks is challenging. A3C intuitively generates multiple policy functions in different sub-agents to explore and generate transaction data for training the model. A sub-agent interacts with the environment within each thread. Fast exploration can be achieved once the transactions among the agents are guaranteed to be different as long as the state of the environment in each thread is different. Using a method called Hogwild, each sub-agent updates the agent's global policy-network asynchronously after collecting transitions and updating its policy function parameters. After sub-agents synchronise with the agent in every thread, they move on to the next iteration. The following experiments suggested that asynchronous updates are not the main determinants of the algorithm's (i.e. A3C) merit. DeepMind group then began to experiment with the non-asynchronous deterministic method A2C. The sub-agents in A2C interact with environments and generate transitions individually, just like in A3C. However, A2C sub-agents stop collecting data when the transition number achieves a pre-set mini-batch size. The agent trains its global network using these collected data and then simultaneously copies its policy-network to the sub-agent.

The Proximal Policy Optimization (PPO) algorithm [91] absorbed the multiple workers' ideas from A3C/A2C and improved the sample utilisation on old mini-batch data. In PPO, π_{old} is an approximate distribution of π . The ideal π could be learned steadily based on π_{old} while the difference between old and new should not be too large. Then, PPO estimates the new distribution by part-sampling from the old distribution π_{old} with π/π_{old} as weights. PPO also offers a simple 'clip' method to control the deviation. While the ratio, that is, the deviation between the current and old policies, would be clipped if it exceeds a certain range. The policy gradient after clipping would be limited to a certain

range so that the neural network parameter updates will not be too outrageous. With the help of clipping, no matter how many steps are updated, there is a clip for the agent to block training bias.

2.2.2 Reinforcement learning in manufacturing system schedule optimisation

The development of RL attracted the attention of operating researchers. By involving the temporal-difference method in NASA space shuttle load handling, Zhang and Dietterich [92,93] were able to outperform an iterative repair method [94]. This case became the first application of using RL to solve a flexible job-shop scheduling problem (FJSP).

Aiming to minimise job tardiness, Riedmiller and Riedmiller [95] used a “multilayer-perceptron neural network”, what we usually call deep neural network (DNN) now, based on Q-learning to solve a global production scheduling problem. This RL agent has proved its effectiveness over heuristic dispatching rules, including first-in-first-out (FIFO), earliest due date (EDD), minimum-slack (MS), longest processing time (LPT), and shortest processing time (SPT), in a single and three-resource production line.

Paternina-Arboleda and Das [96] introduced multi-agent RL to research a single server and multiple products case to obtain a dynamic control policy for stochastic- lot scheduling. The state-action space is about 1.8×10^7 . The proposed decision tree classifiers, which developed from the AlphaGo approach, help RL analyse a complex production system hence gaining higher productivity of such a system. However, the authors also warned that the NN parameters of each agent, including state space partition, are crucial for performance.

Similar to Paternina-Arboleda’s [96] work, Gabel and Riedmiller [97] proposed a job-shop scheduling policy optimisation method with “neural fitted Q iterations”, which is under the Q-learning category. This multi-agent RL approach is partially observable. The experiments suggested that the proposed approach performed much better than basic dispatching rules. In the competition with

sophisticated heuristics-based dispatching rules, the agent can still provide schedules closer to the theoretical optimum.

Compared with flow job-shop scheduling, parallel machine scheduling needs to consider machine task allocation. It is complex to design system status and agent's actions. The parallel machine scheduling optimisation algorithm based on RL focus on dynamic scheduling problems. By asking a Q-learning-based RL agent to select the best dispatching heuristic among the five heuristics methods, Zhang et al. [98] minimised the mean weighted tardiness of a dynamic parallel machine system. The transformed RL scheduling problem considered sequence-dependant setup times and machine-job qualification.

A Q-learning-based RL agent's performance was compared with four heuristic methods (weighted shortest processing time, weighted modified due date, apparent tardiness cost, and weighted cost over time) to minimise the mean-weighted tardiness in Zhang et al. [99]. The authors concluded that all four heuristics failed to generate a superior solution than that provided by the Q-learning-based RL agent. In this n-task m-machine environment, the agent can choose one of five actions among no job or one job picked according to one of the four heuristic methods. The authors also highlighted that even though the RL training may take much longer than heuristics in such a parallel machine setup, applying it to an online environment is still acceptable.

Palombarini and Martínez [100] targeted uncertainty across an entire system (machine breakdown, rush order, resource shortage, reprocessing, and new orders). They proposed a Q-learning-based rescheduling agent trained in a relational MDP-formulated environment. The Q-values are stored in a logical regression tree for compiling a large amount of knowledge about repair policies, hence providing a straightforward reschedule once an interrupting event occurs.

Stricker et al. [101] used Q-learning to optimise an 800-process wafer fabrication system's online scheduling system. The agent generates the schedule by suggesting an order dispatching sequence and machine assignments. The results demonstrated an appealing performance that improved system utilisation by 8% and decreased lead time by 7 minutes.

Another semiconductor industry case from Waschneck et al. [102,103] expanded deep Q-learning to multi-agent scheduling with a continuous-time discrete-event model. Although the researchers did not outperform their highlighted heuristic method Shifting Bottleneck Heuristics (SBHs) [104], the RL approach demonstrated its feasibility in an intelligent multi-agent system with multiple interacting autonomous entities.

Instead of deciding on action for every machine at every step, Q-learning is also being used to optimise a schedule by selecting among multi-dispatching rules. A real-time, at each decision point, flexible manufacturing system scheduling agent proposed by Shiue, Lee, and Su [104] chooses the best multiple dispatching rules combinations among the shortest process time, shortest remaining process time, minimum slack time, shortest imminent operation time, and earliest due date. The result was significantly better than other ones by meta-heuristics approaches.

While the adoption of intelligent production systems and continuously maturing RL technologies has been experiencing an upward trend recently, Zhou et al. [105] developed a double Q-learning-based framework to minimise the maximum completion time of a dynamic scheduling problem in a smart manufacturing environment. Such a system can provide three kinds of tasks across five machines. Moreover, these machines can handle up to two of the three tasks with different capabilities.

Researchers of cutting-edge RL applications prefer skipping the value function and directly training a policy search function network. These policy-based RL algorithms are characterised by low sample utilisation rates, hence falling into local optimisation. However, with their own exploration mechanisms, these policy-based RLs have not yet been widely adopted for scheduling problems.

These typical policy-based algorithms include A3C, which involves multiple workers, Trust Region Policy Optimization (TRPO), which improves the actor-network by only using the function and provides guaranteed monotonic improvement; and Proximal Policy Optimization (PPO), which combines the advantages from both A3C and TRPO.

Rummukainen and Nurminen [106] formulated a stochastic economic lot scheduling problem in a semi-Markov setting. In their 3×10^9 state space environment, PPO provided a promising control policy after 11×10^7 training iterations. The authors also highlighted that PPO2 performs in a more stable way than PPO1 in their experience.

In Kuhnle et al. [107], the authors developed an autonomous dispatching algorithm. They applied it in a semiconductors manufacturing system characterised by small batch sizes and extensive product variety, resulting in complex processes. The authors employed TRPO to train the agent and observed that the trained scheduling agent algorithm outperformed the already-used FIFO rule.

To tackle generic job-shop scheduling problems (JSSP), Zhang et al. [108] represented job-shop instances into disjunctive graphs. These scheduling graphs can therefore be operated by a graph neural network (GNN) based policy. PPO is used to find out a simple but effective priority dispatch rule. Experiments showed that the proposed framework outperformed 4 priority dispatching rules, including shortest processing time, most work remaining (MWKR), most operations remaining, and a minimum ratio of flow due date to most work remaining.

To solve the adverse effect of unexpected machine breakdowns and material problems on the initial job scheduling, Liu et al. [109] formulate a job shop scheduling problem as a sequential decision-making problem. In such a multi-agent environment, the RL agent trained by an actor-critic-based parallel training method combines asynchronous update and deep deterministic policy gradient. The method is quite similar to the proximal policy optimisation (PPO).

Robust and stochastic optimisation has been addressed by Hubbs et al. [110] by applying advantage actor-critic (A2C) to solve a chemical production scheduling. Experiments showed that RL's speed and flexibility outperformed mixed-integer linear programming. Authors suggest that RL helps realise the real-time scheduling system optimisation in a single-stage multi-product environment.

Chen and Tian [111] proposed a novel two-stage RL end-to-end approach for a range of combinatorial optimisation problems. The authors applied their approach to a job-shop scheduling problem with 100,000 random job sequence samples. The experimental results showed the effectiveness of the proposed algorithm on non-uniform job length and non-uniform resource setup.

2.3 Key findings and research gaps

Given the two sections of literature discussion above, several observations about the contemporary literature that addresses RMS scheduling and the corresponding RL-based solutions are made.

First, a crucial deficiency observed in the above-mentioned papers is the absence of the application of state-of-the-art reinforcement learning techniques. This is the primary and most notable gap that requires attention in future research endeavours. It is imperative to explore and utilise the latest advancements in reinforcement learning to enhance the efficacy and accuracy of the research outcomes. Addressing this gap would not only add novelty and sophistication to the research domain but also contribute significantly to the progress and advancement of the field.

Next, most scheduling-based problems, including those within an RMS context, were modelled and solved using purely mathematical models (mathematical programming models). These models, although useful, require making many assumptions and simplifications, particularly regarding stochasticity and dynamism of the modelled system (key attributes of RMS), which can drift the representation of these models from the nature of RMS scheduling problems.

Then, as a specific branch of flexible job-shop problems, RMS scheduling contains more flexibility and complexity that pose real challenge current methods such as compatible module switching, scale expansion, and quality control. Another finding of the literature review is that much of the contemporary literature did not explicitly account for fluctuating markets, which was one of the most important drivers of proposing the RMS paradigm. At the same time, although there is much work on value-based on-line RL, namely DQN, for

solving production scheduling problems, the study of policy-based off-line RL, which is receiving increasing interest from the RL community, for scheduling remains largely unexplored.

To summarise, the overall gap that this research will attempt to fill is:

The lack of on-policy RL-based real-time scheduling algorithms that embrace core characteristics, stochasticity, dynamism, and modularity attributes of RMS.

To respond the research gap above, a framework will be proposed then discussed in detail in the next chapter.

3 Training Framework

3.1 Introduction

This chapter proposes an end-to-end framework to get a reinforcement learning (RL)-based intelligent scheduling policy for a reconfigurable manufacturing system (RMS). The desired function of such a policy is that it can receive a constantly updating demand and then output a production schedule. This framework comprises three components built up from a general RL project procedure [75], as shown in Figure 3-1 below.

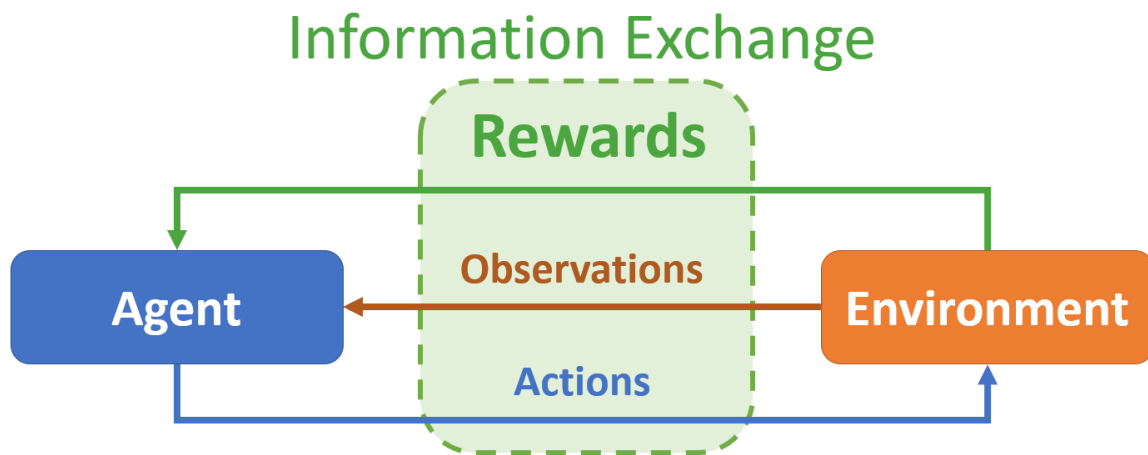


Figure 3-1 General RL agent–environment Interaction

The first major component of an RL project is the environment, which is presented in orange in Figure 3-1. The environment contains the problem that the RL project is aiming to solve. Regardless of the detailed forms, such as grid worlds or physical simulators, these problems should be formulated in a classical sequential decision-making form named finite Markov decision processes (MDPs) [75]. By evaluating the immediate and foreseeing feedback, MDPs choose appropriate actions in different situations. The next critical component is the agent. As the carrier of the desired behaviour patterns, such an agent should have enough artificial intelligence to estimate the value $q^*(s, a)$ of each action a in each state s . The place for researchers to formulate the final expectation and the driver of desired behaviours is the reward function in the information exchange part, as the RL agents shape their behaviour through trial-and-error experiences with the corresponding environments.

The rest of this chapter is organised as follows: first, designing and presenting a discrete-event simulation-based RMS model, whose goal is fulfilling a market simulator which mimics the fluctuating demand. This simplified model contains several subsystems for an agent to control. After the model is built, a reward function is presented for achieving the key performance indicators (e.g., throughput and lead time) of an RMS to an agent, no matter it is based on DQN or PPO, understandable way; different methods to build and train agents describe at last.

3.2 Reconfigurable Manufacturing System Modelling

It can be seen from the aforementioned content that the AI capabilities of an RL agent comes from its interaction with the environment. Therefore, a properly abstracted and simplified environment can simulate the actual system efficiently while providing sufficient data for agent training.

3.2.1 The fluctuating market simulator

The RMS paradigm is proposed to handle the frequently changing external variables, including but not limited to design updates for existing products, demand fluctuation, process improvements, and legitimation or regulation changes [3,112,113]. This research focuses mainly on optimising task sequences in scheduling, without taking into consideration other external constraints that may commonly affect scheduling processes. The scope of this study is limited to exploring the most effective ways of organising and arranging tasks in a schedule, in order to improve efficiency and productivity. External constraints such as resource availability, budgetary constraints, and environmental factors will not be considered in this research, as the aim is to specifically examine the impact of task sequencing on scheduling outcomes. By narrowing the focus to this particular aspect of scheduling, the study aims to provide valuable insights into the optimisation of task sequences and inform future research in this area. This demand generator is the first event generator. As the first event driver in this study, an order generator is proposed to simulate unpredictable demand changes. This order generator is presented as a list, which contains several randomly generated orders to mimic fluctuating demand.

In each order, I_n , requests of one kind of product are generated with a random batch size and a due date. According to the variant limitation of an RMS, the total product variant should not be considered unlimited but a limit integer. Once a deadline is due, this dynamic updating list tries to withdraw the required quantity of a certain product and then replenish a new order. To reflect the market's reality, every order comes with a fixed deadline under the strictest setting - zero tolerance for tardiness. If an order misses the deadline due to insufficient inventory, there will be no extension or delay to fulfil the demand. Figure 3-2 schematically describes the order generator.

RMSs in this research do not possess exact and complete information about incoming orders or jobs during the simulation time. These orders may arrive at an RMS at any time following the Poisson or uniform random distribution.

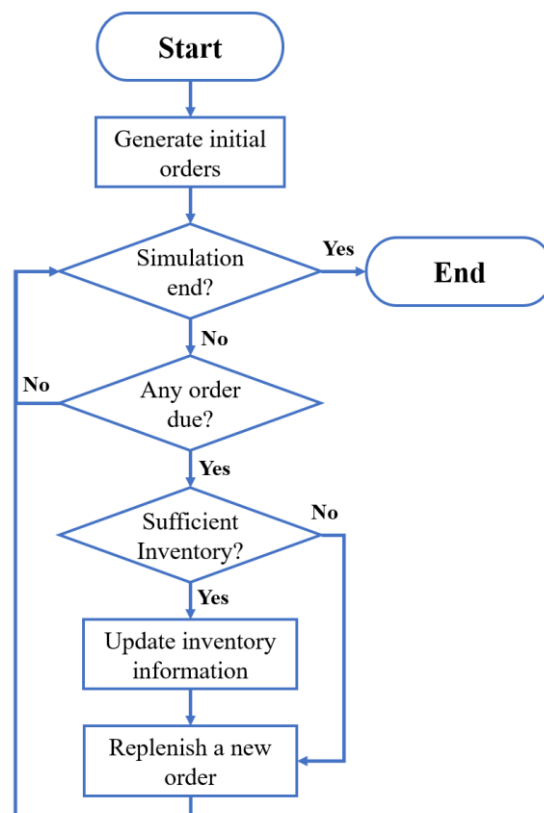


Figure 3-2 A stochastically updating order list with 3 products

3.2.2 The Core Components of an RMS

Scholars [12] have identified six primary attributes, namely modularity, integrability, diagnosability, scalability, customizability, and convertibility, that distinguish reconfigurable manufacturing systems from other manufacturing systems. These essential features pose significant challenges to contemporary scheduling and schedule optimisation techniques, as they introduce complexity and dynamism into the production system, increase the number of variables that must be considered, and require real-time monitoring and feedback.. Production scheduling optimisation can have multiple criteria, such as maximising profit, efficiency, equipment utilisation, on-time delivery, or stock occupancy. Among these six core characteristics, modularity provides fundamental flexibility on the structural level, while convertibility allows quick changeovers between product families. Focused on these two characteristics, an RMS model proposed in this research comprised 3 major physical subsystems, including reconfigurable machine tools (RMTs), a material handling system (MHS), and an inventory system.

RMTs are, first and foremost, a subsystem of an RMS. RMTs provide the only value-adding action inside the system: manufacturing material into work-in-process (WIPs) or finished goods. The second component of an RMS is an MHS which responds to all physical transportation inside an RMS, including delivering a module or material to an RMT and transferring WIPs or finished products from an RMT to the next process. The third and last critical subsystem of an RMS is the storage and inventory system, which includes raw material storage, module storage, WIP storage, and finished product storage.

3.2.2.1 Reconfigurable Machine Tools

One of the most important RMS principles is “*maximise system productivity by reconfiguring operations and reallocating tasks to machines*” [1]. An RMT can only function properly with a module. In this research, an RMT can reconfigure into different setups by consuming a certain amount of time. If an RMT gets a module-match material, it will spend a certain amount of time to implement production operations.

3.2.2.2 Material Handling System

The second subsystem is a Material Handling System (MHS). The duty of a proper MHS is to provide optimal material flow. This research considered two kinds of MHS. The first one is an ideal MHS which is available and on standby all the time to deliver any material, work-in-process (WIP), or RMT modules. The second one is an automated guided vehicle (AGV)-based MHS. This setup exists when an RMS is simplified as a grid-world RMS-DES model, and the AGVs of the MHS can transport freely. During the simulation process, the RL agent would release the job to AGVs, and RMTs react passively with such AGVs. Every individual AGV can do 6 kinds of actions, which include 5 types of movement: up, down, left, right, and staying rest. The rest interacts with other elements within an RMS.

Table 3-1 AGV action space

Up	Down	Left	Right	Rest	Interacting
----	------	------	-------	------	-------------

The first 4 kinds of movement are not always available as the grid-world RMS models have a boundary. When an agent asks an AGV to move toward a wall, the action would be considered as staying rest.

Combining movements and interactions, there are 4 kinds of jobs for AGVs to drive an RMS, including reconfiguring RMTs, delivering raw materials, transporting WIPs, and finished goods. To reconfigure an RMT, a free AGV should first interact with module storage to receive a module. Then the AGV transport this module to a free RMT to replace its original module. Both the AGV and RMT stay locked during the reconfiguring period and ignore all commands from the RL agent. After an RMT finishes its reconfiguration, the AGV should carry the module back to the module storage or another RMT if it needs such a module. The raw material, WIP, and finished goods delivery share a similar pattern. For raw material transport, a free AGV first interacts with a material inventory. Then the AGV moves to a free RMT equipped with a proper module and drops off the material. This RMT would make this raw material into a WIP or a finished good, then need an AGV to move it to an inventory.

All possible events due to AGV and RMS-DES interactions are listed in Table 3-2 below.

Table 3-2 AGV-based MHS events

Event	Event description
1	An AGV moves to an available place.
2	An AGV moves toward a wall, and this AGV stays rested until the next turn.
3	An AGV stays rested for one turn.
4	An AGV with empty cargo attempts to interact with the module storage. RMS will randomly pick one kind of module.
5	An AGV carries a module and attempts to interact with the module storage. RMS will randomly unload the module from the AGV or switch to one kind of module.
6	An AGV carries a non-module part and attempts to interact with the module storage. This AGV would stay rested until the next turn.
7	An AGV with empty cargo attempts to interact with the material storage. RMS will randomly pick one kind of material and load it to the AGV.
8	An AGV carries a material and attempts to interact with the material storage. RMS will switch to another kind of material randomly.
9	An AGV carries a non-material load and attempts to interact with the material storage. This AGV would stay rested until the next turn.
10	An AGV carries a finished good and attempts to interact with the inventory. RMS will unload the finished good from this AGV to the inventory.
11	An AGV with a module or raw material attempts to interact with the inventory. This AGV would stay rested until the next turn.
12	An AGV with empty cargo attempts to interact with the inventory. This AGV would load a WIP randomly or stays rested until the next turn if there is no WIP in the inventory.
13	An AGV with empty cargo attempts to interact with an RMT that has a WIP or finished good. RMS will unload the WIP or finished goods to the AGV and set this RMT to free from the next turn.
14	An AGV carries a material or WIP and attempts to interact with a free RMT with a matched module. RMS will unload the material or WIP and set the RMT to busy mode until it finishes.
15	An AGV carries a material attempts to interact with a free RMT with an unmatched module. This AGV would stay rested until the next turn.
16	An AGV carries a module that attempts to interact with a free RMT with a different module. RMS will switch the module and set this RMT to busy until it finishes reconfiguration.
17	An AGV carries a module that attempts to interact with a free RMT with the same module. This AGV would stay rest until the next turn.

- 18 An AGV attempts to interact with a busy RMT. RMS will freeze this AGV until the next turn and return a small local negative reward.
-

3.2.2.3 Inventory

An inventory subsystem generally describes a place to store modules, raw materials, work-in-process, and finished-good. In this research, an inventory has no limit on its capacity. As keeping the inventory level at a reasonable range is one of the optimising goals, agents are expected to build the inventory management policy themselves. Infinite space should help explore the policy. The optimal scheduling policy can help identify the upper limit of the required inventory space when designing and implementing a real-world RMS.

3.2.2.4 Assumptions Summary

- (i) All RMTs are available at time zero.
- (ii) All RMTs are equipped with one kind of module.
- (iii) One RMT can only process one module-matched operation at a time.
- (iv) The RMSs are considered as discrete manufacturing (v) No precedence constraints as all operations are independent of each other.

3.2.3 Modelling Approach

This paper uses discrete-event simulation (DES) to build virtual RMSs. Considering that some elements inside the RMS model have dynamic interactive relationships, spatial features, and internal strategic behaviours, the proposed RMS-DES model contains some hybridised features from agent-based simulation (ABM) [114]. These features can improve the generality of the virtual system. However, this paper would not use the terminology from ABM content as these terms can be confusing to the readers with RL terms.

Theoretically, an RL agent can solve any environment formed in the Markov decision process. However, the actual RL implementations easily crash into problems due to inexperienced setups, such as sparse reward distribution or a big policy shift due to small parameter updates. Therefore, an efficient environment is the base of a successful RL project. DES is one of three major

systems-simulating approaches, with the rest two being ABM and system dynamics. The activity-oriented feature makes DES simulate the manufacturing system efficiently and steadily. As a powerful tool for dealing with and observing sequential jobs of an entity inside the system, DES is naturally suitable for data-hunger RL projects. In industrial cases [115–118], DES can reveal optimal system performance by adjusting parameters which that field study would be hard to try out. However, RMS scheduling has big challenges in dealing with complex system states and cooperating with every component inside the system.

Though DES has been used for scheduling research for decades [118,119], how to implement such a technique in RL was not discussed thoroughly. The following sections proposed three approaches to designing an RMS-DES. One is order-focused for push production, while the other two are machine-focused pull production.

3.2.3.1 An Order-focused Modelling Approach

The most straightforward goal of training a scheduling agent is to ask it to release jobs at an exact time to exact machine. The first RMS-DES request a specific product or not at every decision-making point.

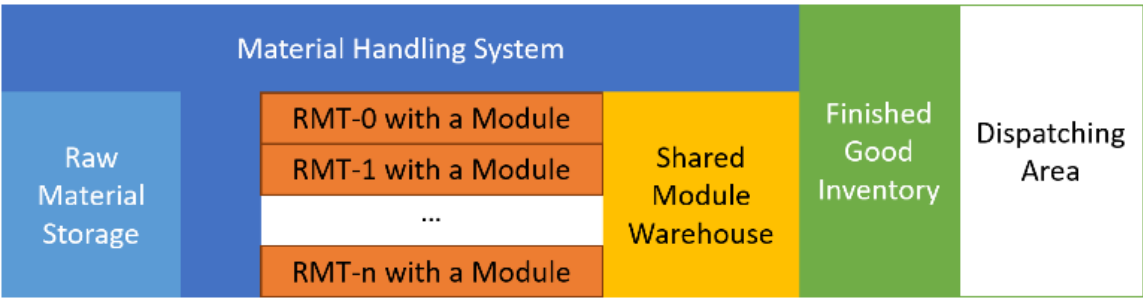


Figure 3-3 A simplified reconfigurable manufacturing system

This dimensionless generic RMS has 4 major components presented in Figure 3-3 above.

The first component of this RMS-DES is raw material storage, where the scheduling agent pushes jobs out.

Second, after receiving the material dispatching order, a material handling system (MHS) will spend some time transporting one assigned material to the assigned RMT. This MHS is considered with infinite availability, which means the transporting time from raw material storage to the assigned destination is constant.

The third component is a group of parallel RMTs with a shared module warehouse—these RMTs respond to manufacturing products from material to product. Unique products need different RMT modules and production times. Suppose an RMT receives a raw material current configuration cannot make. In that case, it will ask the module warehouse to send the 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. The finished goods transport time is considered constant in the same setting as the raw material moving process. After a finished good arrival, the inventory will check whether the current stock can fulfil the first required order. The inventory will pile up the stock until the stock level is sufficient.

3.2.3.2 Machine-focused Modelling Approaches

The first machine-focused RMS-DES is a grid-world shape model presented in

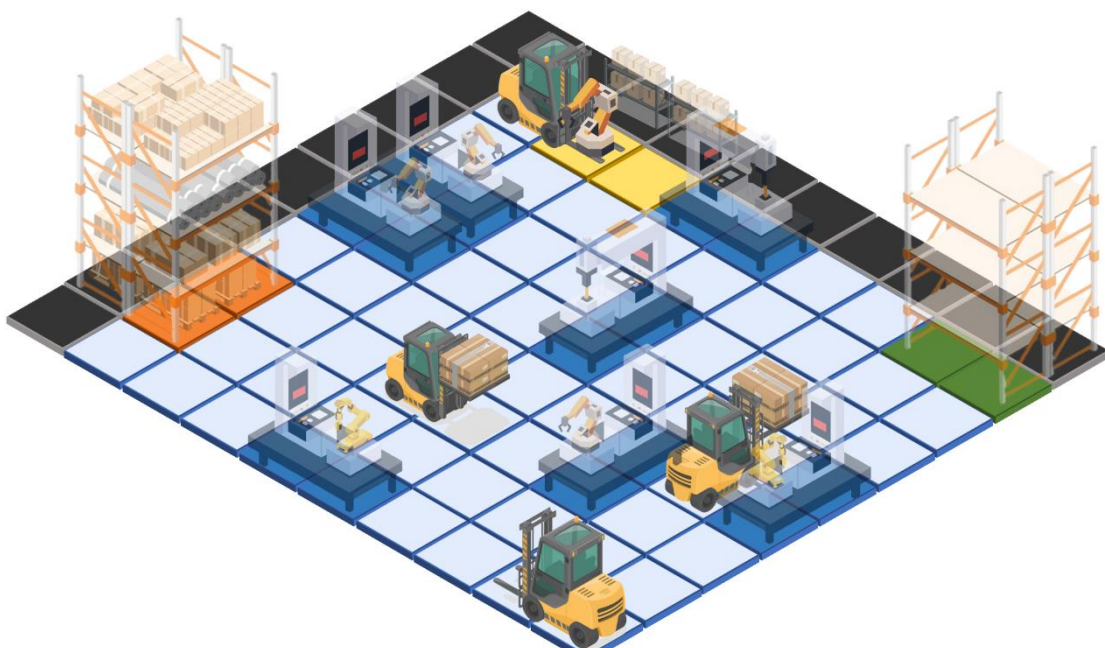


Figure 3-4 A Grid-world Shape RMS-DES

Figure 3-4 below.

In this simplified $m \times n$ grid-world-shaped RMS, there are several Automated Guided Vehicles (AGVs) for the RL agent to assign jobs. Several reconfigurable Machine Tools (RMT) are located in blue blocks across the grid-world.

As walls surround this environment at the edges, AGVs, present as forklift icons with randomly assigned locations when initialising, are not allowed to drive in. There is no AGV location interference inside the boundaries, present as two forklift in the same block.

All RMTs are fixed in place after initialisation, and the AGVs can move “freely”. During simulation, RMTs react passively with AGV interactions with several rules according to Table 3-2, listed in the previous section.

The second machine-focused approach shifts the focus point from the MHS to RMTs. It allows the agent to request a special product or not from every individual RMT directly. In such a case, MHS reacts passively to the pulled orders from RMTs. If an RMT decides to produce a certain product, the MHS will check whether that RMT is equipped with a proper module. The MHS will send a module for the RMT to reconfigure if the existing module cannot produce the requested product. MHS would deliver raw material to an RMT with a matched module. Once the production process finishes, the finished goods will be transported to inventory immediately. With such assumptions, all material moving times are considered constant.

3.3 Reward Function for Multiple Objectives

Defining a proper structure to formalise objectives into a reward function is one of the most distinctive features of reinforcement learning. Through the environment, the *reward* is the special signal which is categorised for desired goals. The reward signal is presented at each moment as a single scalar value. Informally, the intelligence of an RL agent is measured by its capability to maximise the total received reward. This means that the instant reward needs to be maximised, not the cumulative reward over time. Therefore, all criteria

should be reduced to help to maximise the probability expectation of the cumulative sum of reward received by the agent.

As RL agents do not explore the hidden structures inside environments, the training process of an RL agent is always learning how to maximise the reward. For desired behaviours, providing a positive reward helps an agent maximise its reward while also achieving the objective. It is therefore crucial that the way the reward has been set up indicates the objective. In particular, reward signals do not impart a priori knowledge of how an agent should achieve its goal. For example, a chess game-playing agent is only rewarded when it wins at the terminal stage but not when it achieves some subgoal, such as capturing a bishop or controlling the central region. The reason behind this is that if achieving such subgoals also gains rewards, the agent might find some way to focus on these subgoals only, hence bypassing the major goal. However, for complex environments, only providing sparse rewards at the terminal step makes agents have to spend much time exploring while having little chance to gain rewards. The following section proposed the reward function to help train an RL in this study, which tries to balance among different scheduling objectives.

3.3.1 Sparse Rewards and Dense Rewards

Most of the time, during training, an agent can only occasionally receive a reward, such as delivering or missing an order. These kinds of rewards in RL content are called *sparse* rewards. Training an agent successfully would be extremely difficult if the reward distribution is sparse. The case study 2 in the next section crashed into this kind of problem. Sparse rewards bring difficulty in 2 different ways mathematically.

The probability is the first difficulty brought by sparse rewards. Both algorithms used in this study, DQN or PPO, are model-free. Starting training is like putting an agent in a maze filled with mist. The way an agent understands the environment is like wandering in the mist carefully. A sparse reward is like a malfunctioned beacon that can only be lightened up until the agent achieves it.

If a sparse reward hides at the far end of a complicated MDP, they would be very little chance an agent can find a correct trajectory to achieve that beacon.

Sparse reward challenge agents' expectation as well. A discount factor $\gamma \in [0, 1]$ is one of the key hyperparameters of an RL project. This factor helps an agent balance an immediate or long-term reward. If the discount factor equals zero, a myopic agent only considers immediate rewards. If the discount factor equals 1, it means an agent calculates the all-around reward. However, an equal to 1 discount factor does not encourage an agent to optimise its performance, like chasing a shorter trajectory. Accordingly, a discount factor is usually set to a value very close to one. In such cases, sparse rewards in a complex system may be worth very little at a state far away from the desired state. The agent would consider that beacon is not worthy of going as the light has been weakened so much by the discount factor.

Two reward-shaping approaches are involved in this study. The first and straightforward one is amplifying the sparse rewards during simulation, as the agent can receive clearer signals. The second one is designing more *dense* rewards to help guide the training. These immediate rewards do not have to receive from the environment directly. For example, StarCraft II Learning Environment researchers used a comprehensive immediate reward function named "Blizzard score" for agents to accumulate, although the real reward is a ternary value.

3.3.2 From Key Performance Indicators to Rewards

In this study, a reward function should guide an agent to deal with fluctuating market demand first and then optimise the operation cost. Following this principle, the rewards generated by different events can be categorised into 3 kinds: positive, neutral, or negative.

The first positive reward is successfully delivering an order. As this kind of event rarely happens, agents gain the largest reward signals from such sparse events. As an order can only be delivered if there is sufficient stock, this research proposes a dense reward when any finished goods or WIP is made.

The immediate rewards for most of the actions are neutral. The influence of actions mostly reflects on opportunity cost or gain for a future event. For example, in an MHS-centred environment, an RL agent should decide whether an AGV should move, stay, or interact. These actions cannot directly generate influence on the stock level, so the agent would receive neutral rewards.

Forbidden actions and objective-related events are two negative reward sources. In a grid-world RMS-DES, AGVs may move toward a wall. Negative rewards and the unchanged system can inform the agent to avoid such actions. The inventory level is the counterpart of a dense positive reward for producing the product. In this study, the inventory level is a constantly checking variable for an agent. Comparing the frequency of inventory checking and product making, the reward due to inventory level is even denser.

3.4 An Agent Can Schedule

From superhuman game performance to cutting-edge nuclear fusion research, the field of RL has literally exploded in the current decade. Deep RL is one of many ways to learn intelligent behaviour in complex dynamic environments. The core intelligence of an RL agent contains in an artificial neural network π with parameter θ . Like other machine learning problems, there are three steps to training artificial intelligence: finding an unknown function, defining a loss function, and optimising. Any RL project consists of three components: an actor, an environment, and a reward function that links the actor and environment.

As this study explores the application of reinforcement learning to optimise the scheduling of an RMS, an intelligent scheduler acts as the agent, while the discrete event simulation model of the RMS serves as the environment. The reward function is based on the system's productivity. By leveraging reinforcement learning, the intelligent scheduler should make informed decisions to achieve optimal scheduling performance, leading to increased efficiency and effectiveness in the manufacturing process.

The π_θ of this study, a schedule policy in aforementioned environments can be considered as an unknown function $f(RMS, Demand) \Rightarrow Scheduling$ where *RMS* and *Demand* are the observation of market demand and system information which represented as a vector or a matrix. This policy network requests the vector or matrix-formed observation from the environment as input. Every neuron in the output layer of the policy network represents a detailed action for manipulating the environments. The series of actions combine to form a schedule.

For the loss function, in supervised learning, the ‘agent’ can get a lot of labels from the input. Even self-supervised learning can only eliminate expensive human labelling work but cannot get rid of the label itself. Auto-encoder, under the unsupervised learning category, shares a similar concept but generates labels by itself. When a human cannot calculate the best solution for a question, in another way, cannot label the solution, reinforcement learning may help in such a place. In the context of reinforcement learning, optimal solutions may not be readily apparent to human agents. Nevertheless, the efficacy of a given solution can be evaluated based on its ability to yield desirable outcomes. A similar scenario in daily life is that customer can easily judge whether a dish taste good or not but hard to provide the cook with an accurate improved proposal. This kind of judgement is the loss function of an RL project.

With the training process is going on, an agent will understand the environment better and hence predict more accurate and optimised state transition and reward.

The following section presents two algorithms to train the policy π_θ . The first one is an off-policy approach represented by the classic deep Q network approach, and a more resilient proximal policy optimisation on-policy represents the second one.

3.4.1 An Off-policy approach

As the literature review section suggested, the deep Q network (DQN) algorithm is a classical off-policy RL approach.

DQN is considered a powerful tool for finding optimal choices from given states and rewards. Figure 3-5 schematically presents how to bridge the environment, reward, and the DQN agent itself.

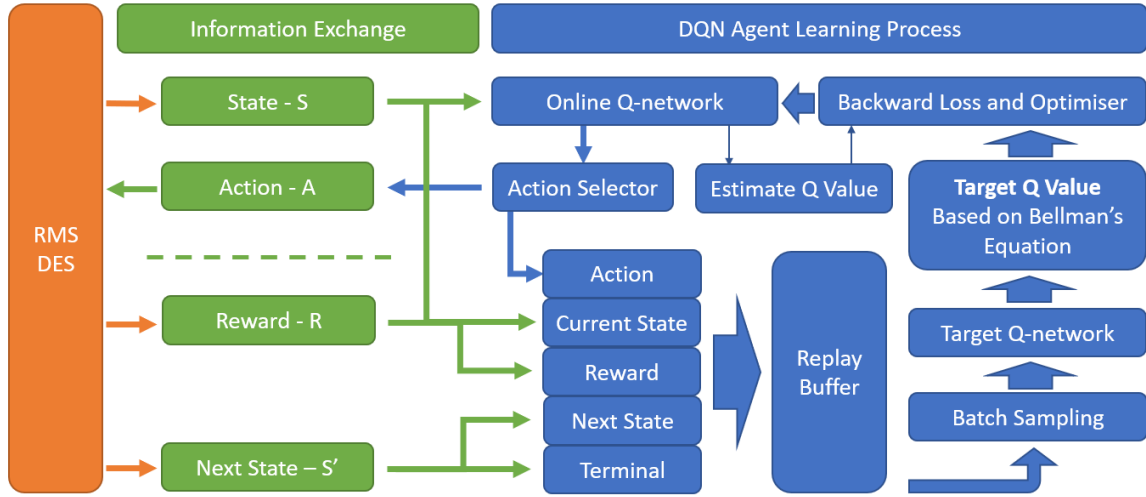


Figure 3-5 Deep Q Network Learning Process

A size-limited replay buffer records tuples which contain rewards and state-transition information feedbacked from DES. All DQNs are trained from the samples from this buffer. This replay buffer has two kinds of storing and retrieving. The first one stores the transitions sequentially, while the other calculates the priority of each transition according to the difference between prediction and actual return and then stores the data within a tree-like database.

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

$$Q(s, a) = r + \gamma \argmax_{a'} Q(s', a') \quad (3-1)$$

Where r is the immediate reward for transmitting from the current state, s , to the next state, s' . A discount factor, $\gamma \in (0, 1]$, weakens and balances the reward from future reward. Since DQN's first introduction, there has been no significant change in the basic training process, as 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 are 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 as 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

$$Q(s, a; \theta) = r + \gamma \max_{a'} Q(s', a'; \theta') \quad (3-2)$$

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

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

The replay memory D , with priority or not, collecting N previous experience and a deep neural network with random weights θ that represent the action-value function Q are initialised at the beginning of a training process. An agent randomly chooses an action until the N length replay memory is fulfilled. Then scheduling agent selects and executes actions according to an ϵ -greedy policy based on Q_θ . ϵ is a hyperparameter representing the probability that the action would be chosen randomly regardless the Q_θ , and it will decrease until a set value as training progresses. Uniform random sampling could get a lot of useless data when storing and sampling transitions to and from a prioritised experience replay (PER). However, only picking up the 'useful' data could also lead to over-fitting. Stochastic prioritisation sampling is used. The probability of a certain transition being sampled can be presented as $P(i) = p_i^\alpha / \sum_k p_k^\alpha$ Where k is the ratio between total transitions in PER and the mini-batch needed to be sampled, and α is a hyperparameter to control how much prioritisation is used. To control the bias brought by stochastic prioritisation sampling, an importance-sampling weight is introduced as: $w_i = (N \cdot P(i))^{-\beta}$ Where β is an incrementing value from β_0 , another hyperparameter to 1. Combining the

importance-sampling weight w with TD error δ , Q-network can be updated and annealed from bias.

Table 3-3 Basic DQL Process of a Scheduling Agent

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 do
04	Observe the initial state s_0
05	for $t = 1$ to T (Terminal state of DES), do
06	Choose an action, a_t , at according to Q_θ with ε greedy probability for a random action
07	execute at on DES and get a 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 the current episode terminates at step $j + 1$
	$r_j + \gamma Q(s_j + 1, \argmax_{a'} Q(s_j + 1, a', \theta), \theta)$ if the current episode does not terminate at step $j + 1$
11	Perform a gradient descent step base on $(y_j - Q(s_j + 1, a_j, \theta))^2$
12	Decrease ε until a set minimum value
13	end for
14	end for

The fundamental training algorithm for Deep Q-Learning (DQL) utilising a solitary neural network has been observed to occasionally overestimate the actual rewards, leading to what is known as maximisation bias. However, decoupling the estimation of action values from the selection of actions can effectively resolve this issue. The main neural network would decide not only the current state's optimal action but also the optimal action for the next state. This approach is known as the Double DQN algorithm, and its Bellman's equation is written as

$$\begin{cases} Q(s, a; \theta) = r + \gamma Q(s', a'; \theta') \\ a' = \operatorname{argmax}_{a'} Q(s', a'; \theta) \end{cases} \quad (3-4)$$

Two same structure network has different parameters θ and θ' . $Q(\theta)$ responds for action selection while $Q(\theta')$ responds for calculating the target. When θ' gets an update from θ every fixed-length period, two networks are built, and the target becomes stabler.

For the dueling DQN, the Q value is made up of two functions, V and A . The equation can be written as $Q(s, a) = V(s) + A(s, a)$. V stands for the state value function of the current state, which tells the expected reward from the current state only. A represents the advantage function, which tells the difference an optimal action can bring and compares it to the other actions. In neural network training, V and A are unidentifiable and cannot simply sum up for a Q estimation. To solve this and chase a better agent training performance, Wang [120] introduced Dueling DQN architecture, which forces the highest Q -value equal to V hence highest A has to be zero. To get optimised 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.

In practice, the dueling DQN is very easy to implement. The basic DQN structure in this research is shown in Figure 3-6 below.

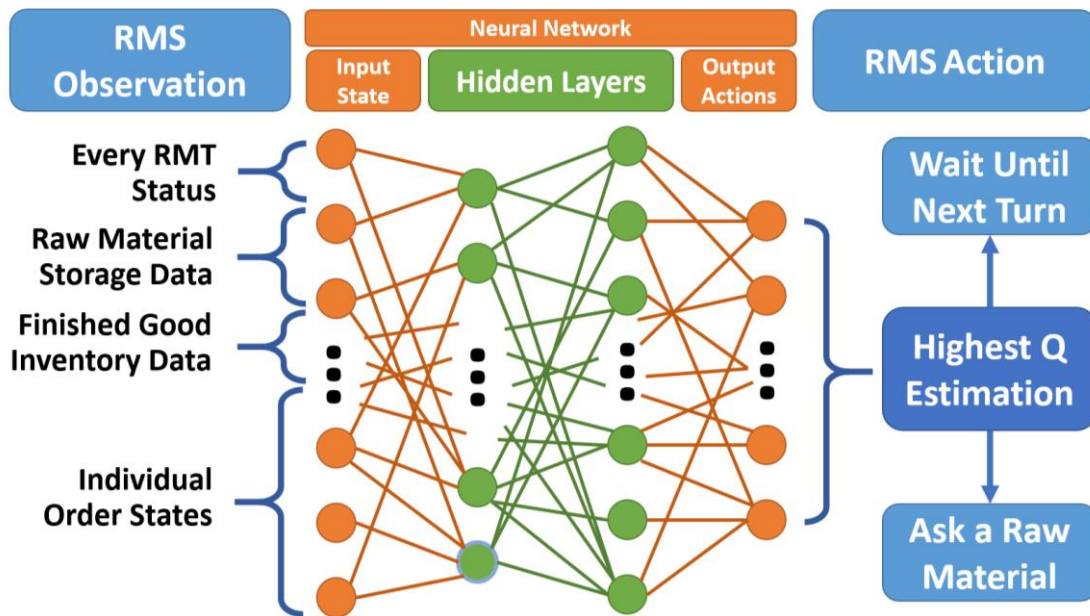


Figure 3-6 Basic DQN for RMS-DES Scheduling

A Q network present in the orange section takes RMS observations as input, then forward calculate through several hidden layers. The output layer contains neurons whose quantity matches the number of actions.

The dueling DQN makes only two differences compared to the basic DQN. The first is that dueling DQN outputs one extra value after the hidden layers to represent the state value V . The original neurons represent Q -value of each action and now become the advantage value A of each action. This leads to the second difference that the calculation of the true Q -value of each action can now be re-written as

$$Q(s, a) = V(s) + (A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a')) \quad (3-5)$$

Summarise, double DQN improves the target calculation and parameter update method, while dueling DQN improves the structure of the neural network itself. These two techniques are conflict-free. Combined with the aforementioned efficient technique, the off-policy approach this study uses is a dueling double deep Q network with prioritised experience replay (DDDQN-PER).

3.4.2 An On-policy approach

As the DQN approach uses the ϵ -greedy method to ensure an agent keeps exploring the environment, on-policy approaches take action based on the probability, which are the normalised scores of each action. The environment-agent interactions can be presented as a trajectory $\tau = \{a_1, s_1, a_2, s_2, a_3, s_3, \dots, a_n, s_n\}$. The policy of choosing a specific trajectory is considered as a product of a series of probability $p_\theta(\tau) = p(s_1)p_\theta(a_1|s_1)p(s_2|s_1, a_1)p_\theta(a_2|s_2)p(s_3|s_2, a_2) \dots$

$$= p(s_1) \prod_{t=1}^T p_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$$

The policy parameter θ is what the agent can control and be trained. The expected reward of one trajectory can now be written as: $\bar{R}_\theta = \sum_\tau R(\tau)p_\theta(\tau) = E_{\tau \sim p_\theta(\tau)}[R(\tau)]$. To use gradient-descend for maximising the expected reward,

the vector differential of $\bar{R}_\theta$ can be written as $\nabla \bar{R}_\theta = \sum_\tau R(\tau) \nabla p_\theta(\tau)$ because $R(\tau)$ does not have to be differentiable. Then change the expected reward to a logarithm form and get $\nabla \bar{R}_\theta = \sum_\tau R(\tau) p_\theta(\tau) \frac{\nabla p_\theta(\tau)}{p_\theta(\tau)} = \sum_\tau R(\tau) p_\theta(\tau) \nabla \log p_\theta(\tau)$. Combining the summation over τ and weighted $p_\theta(\tau)$ then transform $\nabla \bar{R}_\theta$ to an expected form $\nabla \bar{R}_\theta = E_{\tau \sim p_\theta(\tau)} [R(\tau) \nabla \log p_\theta(\tau)]$. The expected values cannot be calculated, so a sampling estimation form makes $\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^N R(\tau^n) \nabla \log p_\theta(\tau) = \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_\theta(a_t^n | s_t^n)$. The final concept for training becomes explicit when doing an action a_t^n at s_t^n helps increase the return, the probability of picking action a_t^n at s_t^n should increase and vice versa. Combine with a learning rate α , Policy Gradient (PG) can now update its policy parameter $\theta \leftarrow \theta + \alpha \nabla \bar{R}_\theta$.

PG has allowed deep neural networks to make significant progress in control tasks like console games, 3D motion, and Go. However, achieving excellent results with PG and its extended methods is, in some ways, difficult because of iterative steps sensitivity. If the iteration step is too small, the training process can be slow, while a huge number can lead the feedback signal to drown in noise hence a performance avalanche. At the same time, PG methods use samples of the interactions only once, which means they are significantly less efficient. This inefficient sampling feature is due to its on-policy nature, which requires millions and billions of total iterations to learn simple tasks. Unlike supervised learning, which is based on a steady and clear labelled data set, stochastic gradient descent (SGD) cannot guarantee a decent local optimum in RL. Thus, the training data for RL are generated being highly dependent on current policy and “burnt after training”. The constantly changing data distribution of observation and rewards leads to instability during the agent training process. RL training is also highly sensitive to hyperparameter tuning, such as initialisation [91]. One slight change in policy can have a great impact on performance.

To address these problems, researchers generalised policy optimisation methods' PG loss, $L^{PG}(\theta) = \hat{E}_t [\log \pi_\theta(a_t | s_t) \hat{A}_t]$, 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 taking the state of the system, s_t , as input and output an action, a_t . An advantage, $\hat{A}_t$, is an estimation of the relative value of the selected action in s_t . The advantage idea is borrowed from off-policy methods to provide an indicator for current state-action pairs. To calculate $\hat{A}_t$, the sum of discounted rewards and a value-function estimation are needed. The discounted reward is a weighted sum of all the rewards that happen after taking the 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.

PPO algorithm is developed for minimising the cost function by balancing among small policy updates, ease of implementation, sample complexity, and ease of hyperparameter tuning. PPO is another PG method variant. In PPO, the value function is presented as a neural network. Using the state and discounted reward records, the value function network can be updated like a supervised learning problem. In this case, the advantage estimation can be presented as discounted rewards, a certain number, subtract the value estimation, and a neural network estimation. The advantage can then instruct the agent whether the actions taken returned a better or worse result than expected. Accordingly, 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, suppose an agent keeps running gradient descent on the same batch of collected rollouts. In that case, the neural network would be updated to far outside the reasonable range as the advantage itself is a noisy estimation.

PPO brought the “Trust Region” concept from Trust Region Policy Optimization (TRPO), another iterative policy optimisation approach that never updates 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 $L(\theta) = \hat{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right]$. 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 a “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 the 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 a relatively conservative form as below.

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon)\hat{A}_t)] \quad (3-6)$$

An expectation operator takes the minimum of either the normal policy gradient's objective $r_t(\theta)\hat{A}_t$, which pushes 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 is 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 to ensure keeping agent exploring, and the final training objective can now be written as shown below.

$$L_t^{PPO}(\theta) = \hat{E}_t[L^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \quad (3-7)$$

Combining the simulation environment and agent training process, the PPO-based framework for training a scheduling agent can be presented as a pseudo-code, as shown in Table 3-4 below.

Table 3-4 Clipped PPO-based Scheduling Agent Training Scheme

-
- | | |
|----|---|
| 01 | 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 |
| 02 | Initialise a job-releasing agent with a random scheduling policy π with parameter θ |
| 03 | for iteration = 1 to limit M do |
| 04 | for actor = 1 to N do |
| 05 | for time $t = 1$ to simulation limit T do |

```

06         Form an observation  $s_t$  from all known orders and their deadlines,
           inventory levels, and RMT states
07         Request an action list  $a_t$  which contains individual commands for
           every RMT
08         Simulate 1 timestep according to  $a_t$ 
09         Form another observation  $s_{t+1}$  based on the same method as  $s_t$ 
10         Calculate the reward  $r_t$  based on  $s_t$ ,  $a_t$ , and  $s_{t+1}$ 
11         Form a transition  $\{s_{t+1}, r_t | s_t, a_t\}$ 
12     end for
13     Calculate advantage estimates  $\hat{A}_1, \dots, \hat{A}_T$  for all transitions and store
           to a rollout buffer
14 end for
15 for epoch = 1 to  $K$  do
16     Optimise the loss objective  $L^{ppo}(\theta)$  by sampling a minibatch of size  $\leq$ 
            $NT$ 
17 end for
18 end for

```

3.5 Other Advanced Tricks for Training

This section lists all causal performance-improving tricks which are not related to building DES models, shaping reward functions, or training algorithms.

3.5.1 OpenAI gym Interface

OpenAI *gym* is an API library that provides several environments with a unified interface, namely *Env*. The *Env* requires models to provide 4 basic components, including: action space description, observation space description, a step function, and a reset function.

Action space is a collection of actions allowed to be operated in the *Env*, whether they are discrete actions, continuous actions, or a mixture of both. In this paper, actions space includes any individual released job, movement of an AGV, or reconfiguring an RMT.

Observation space should present the relevant information provided to the agent by the environment at each time stamp. An observation can be as simple as a few numbers or as complex as many high-dimensional images. The observation space can also be discrete, such as a Boolean state of a light bulb switch.

A step function performs an action on the *Env*, then gets the new state and the corresponding reward, and an indicator shows whether the current simulation is over or not. The step function is the most important interface for an agent as it wraps the model and the reward function together.

A reset function ensures the model can return to its initial state and start a new training again.

3.5.2 Multi-environment & multi-processing

One of the major natural weaknesses of the on-policy algorithm is that the actor updates every training step, and the old interaction becomes obsolete. Using the same policy to interact can significantly improve training efficiency and performance resilience. Hardware utilisation can also be greatly increased as environments with different setups run parallelly on a multi-core processor.

3.5.3 Hyperparameter Tuning and its Automation

Unfortunately, the hyperparameter tuning of RL algorithms requires much more effort compared to supervised learning. Parameters of the reward function, network structure, and training setup can influence the converging speed quite heavily. The common and straightforward industrial solution is hardware expansion. Although a ground-breaking RL AI may need “only” hundreds of CPUs and GPUs [68], It is quite common to apply thousands of CPUs and GPUs for AI projects [121–124].

This research tried a relatively budget-friendly method, OPTUNA [125], to try different hyperparameters before training an agent.

3.5.4 Action Mask

The traditional way to train an RL agent NOT to do something is setting an extremely high NEGATIVE reward. From a simple grid-world application to a much more complex robot-arm control project [25,71], *impossible action = huge drawback* is a default setting for every RL researcher. The same setting is used in the previously proposed framework.

The action mask tries to provide a different approach [126]. As the DES in this research only receives discrete actions, implementing action-masks is not tricky. While an RMT is reconfiguring or producing, any action order should have zero response. As the system won't respond, the reward feedback to the agent could have two forms, zero or negative, for faulty actions. Action masking provides a different way of training: forced-manipulating the probability of choosing an invalid action to zero.

In case study 2 in next chapter, the agent quickly learned to avoid crashing into walls. However, the agent is also "afraid" to explore the environment. As PPO samples the action based on probability, a properly designed action mask embedded in the environment should significantly improve the training speed. The last case study in the next section tried applying an action mask.

4 Case Studies

4.1 Case One: Agent Validation on a dimensionless RMS

As the RMS paradigm is developed for handling order fluctuation, it is crucial to validate whether an RL agent can be trained in an environment with limited stochastic variables. The first case study [127] tried to validate the proposed framework in an environment with clear expectations.

This case focused on minimising the total production time with a deterministic group of 20 individual sequential sub-orders labelled with assigned product variants and quantity and without a deadline. This fixed-length order setting is against the aforementioned fluctuating order generator. The season behind this is validating that an RL agent can solve the RMS scheduling problem in a conventional FJSP problem setup expect reconfiguration. Instead of using the dynamically updating order generator proposed in the framework, a fixed order list has two advantages. First, the limited number of orders can help identify an optimal solution. RL projects normally cannot have a clear optimal solution as their environment are generally highly dynamic and full of internal interactions. By limiting the dynamic feature of the environment, the researchers can have a clear performance baseline and do further performance among the optimal solution, RL-based policy, random acting, and conventional dispatching rules. The second advantage of using a stable order list is to reduce the uncertainties in the training process. As a result, the training process of different kinds of agents can be presented more explicitly—the order list details are shown in Table 4-1 below.

Table 4-1 Case Study One Order List: Total 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

Three dimensionless RMS-DES, presented in the previous chapter, based on the same structure, are studied in different experiments in this case to fulfil this virtual order list.

An RMS can contain 1 to 3 RMTs. Each RMT can equip two kinds of modules to produce product variants 1 and 2. These RMTs need to reconfigure between the two modules to manufacture a specific product variant. The reconfiguration time for product variants 0 and 1 are identical, 300 minutes in all numerical cases.

After an RMT reconfigures to the correct form, the cycle time for producing one finished good is 1 min. All finished goods will be temporally stocked in the inventory until the quantity fulfils the first unfinished order.



Figure 4-1 RMS02 ideal Schedule with Dual Product-0 Configuration

To fulfil these 166 product 0 and 362 product 1, all three RMS with different RMT numbers need to raise up to 9 reconfigurations when it follows FIFO. As these orders have no deadline, the optimal solution for the RMS with one RMT, RMS01, is to reconfigure 1 time and spend 828 minutes to finish all orders. The optimal policy for the RMS with two RMTs, RMS02, depends on the initialising situations. If both RMTs start with a module for producing product-0, the optimal solution is for one RMT to start reconfiguring itself for producing product-1 without hesitation while the other RMT starts producing Product-0. When demand for Product-0 is completely fulfilled after 166 production times, this RMT should start reconfiguring itself for producing Product-1. The total

makespan, in this case, is 564 mins. The ideal schedule presents schematically in Figure 4-1 in the previous page.

If RMS02 starts with RMTs with different configurations, the ideal total makespan should be 362 minutes as the RMT with Product-0 configuration has no sufficient time to reconfigure itself and produce Product-1. On the other hand, if RMS02 starts with RMTs in Product-1 configuration, the ideal total makespan is 366 minutes if one of two RMTs starts reconfiguring itself immediately.

With 3 individual RMTs, RMS03 have 4 different initial states. Therefore, when all RMTs start in Product-0 mode, the ideal schedule shows in Figure 4-2 below.

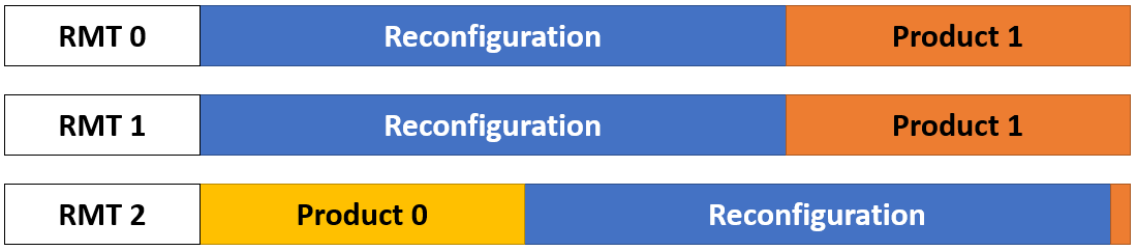


Figure 4-2 The RMS03 ideal schedule with 3 RMTs in Product-0 configuration

Two of three RMTs should start reconfiguring to product-1 immediately, while the remaining one needs reconfiguration after finishing all Product-0 orders. The total makespan is 476 minutes.

When RMS03 start with one or two RMTs in Product-0 mode, the ideal schedule shows in Figure 4-3 and Figure 4-4 below.

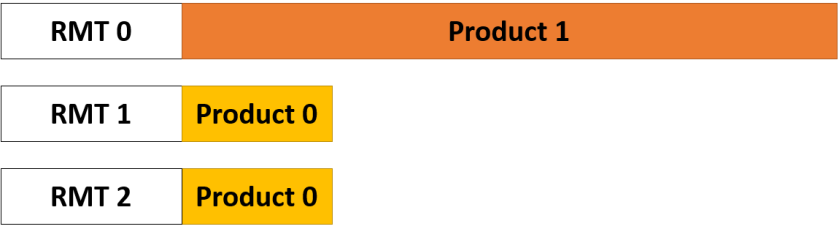


Figure 4-3 The RMS03 ideal schedule with 2 RMTs in Product-0 Configuration

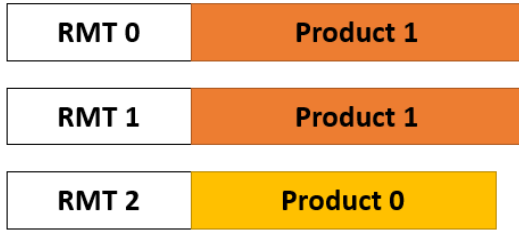


Figure 4-4 The RMS03 ideal schedule with 2 RMTs in Product-1 configuration

All RMTs should start producing the module-matched products immediately and involve no reconfiguration. The ideal total makespan of two Product-0 RMT setups is 366 minutes, and the ideal makespan for two Product-1 RMTs setups is 183 minutes.

When all RMTs start in Product-1 mode, the ideal schedule shows in Figure 4-5 below.

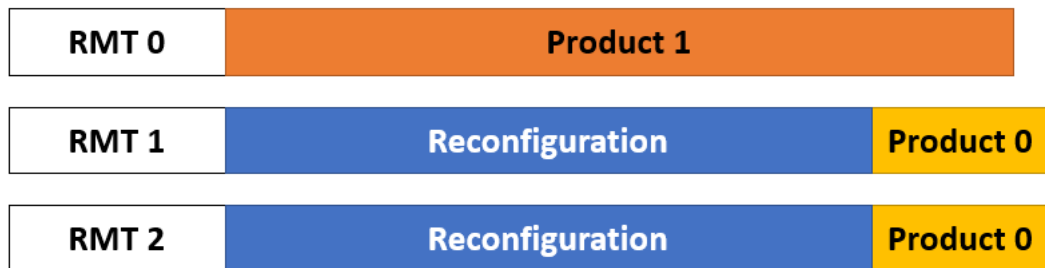


Figure 4-5 The RMS03 ideal schedule with 3 RMTs in Product-1 configuration

To achieve the 383 minutes ideal makespan, two RMTs should start reconfiguration after the simulation start.

Aiming at the shortest total production time and minimum reconfiguration times, the reward function for these RMS-DES models presents in Table 4-2 below.

Table 4-2 Case Study One Reward Function

Scenario	Reward Items	Reward Generating Rate
1	A raw material stays in a storage	0
2	An RMT stays idle	0
3	An RMT is manufacturing a raw material	+5

4	An RMT is reconfiguring itself	-50
5	RMT is busy, but the agent decides to send in a material	-100
6	A finished good dispatched with an order	+100
7	All order cleared	+1,000,000

Scenario 1-6 is the dense rewards check every time step through all simulations, while the 7th is a special one for termination.

In this case, the RMS-DES models run on a 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 encourages an agent to keep an RMS in a value-adding state instead of constantly reconfiguring. While an agent implements a logy scheduling policy, there will be no positive or negative reward. When RMT reconfigures itself, it generates a -50 reward per unit time. When an RMT raw material is in manufacturing, it will generate a +5 reward per unit time until the process is finished and the goods send to storage. One finished good in storage generates -1 reward per unit time. The agent will receive a -100 reward if it decides to send material to a busy RMT.

Three kinds of scenarios will end the current training episode to avoid unnecessary computational power waste.

In every simulation step, RMS-DES first updates its current state and calculates 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 the current simulation episode performed too badly to the end. Suppose the accumulated reward does not exceed the threshold (set as negative 2 million in all numerical cases in this paper). In that case, DES will keep running until a pre-set simulation time runs out (set as 7200 mins which is equal to one week), or all orders are fulfilled.

It requires unique DQN aggregate output layers. RMS01 with only one RMT has 3 possible discrete actions: produce a product, reconfigure with the other module, and keep an RMT idle. For an RMS that contains more than one RMT, the action space expands exponentially due to the increasing number of RMTs. RMS02 contains 2 RMTs has $3^2 = 9$ kinds of potential action combinations. RMS03 with 3 RMTs has $3^3 = 27$ kinds of possible reaction modes.

To choose the optimal solution in the current policy, the agent needs to receive an observation. This input observation consists of four kinds of features, listed in Table X below, which include every RMT state, remaining raw material state, finished good stock state, and order state. The state of an RMT includes three elements which are installed module type, remaining time to finish reconfiguration T_{cfg} , and remaining time to finish the current production T_{pdc} . The quantity of each kind of raw material still in storage makes 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 finished good state describes the quantity of finished product for each kind of product.

Table 4-3 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

4.1.1 Experiment 1-1: Basic DQN Algorithm Validation

The first experiment trained two basic DQN agents for RMS01 and RMS02. Agents will be trained one iteration when it gets one state-transition tuple.

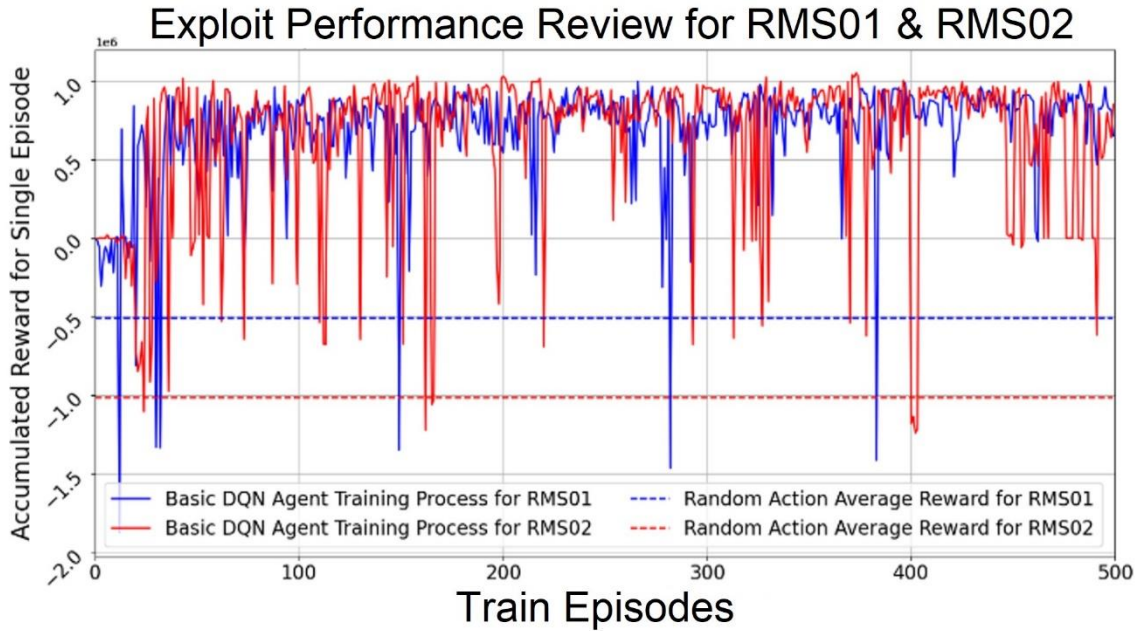


Figure 4-6 Basic DQN Training Process Comparison (RMS01&02)

Simulation should first fill a replay buffer which can store 100,000 training transitions. Every train iteration randomly retrieves 3000 samples; in another way, the mini-batch size for each gradient update equals 3000 from the replay buffer. Every 500 iterations, consider one training episode. After finishing each training episode, the agent runs one simulation in exploit mode to benchmark its performance using an accumulated reward. Two solid lines in Figure 4-6 present the benchmark trends of two agents of RMS01 and RMS02. Dashed lines are the average accumulated rewards when RMS01 and RMS02 act randomly. Figure 4-6 also illustrates that both agents converged quickly after only around 30 turns. However, increasing training episodes did not bring more advantages to agents.

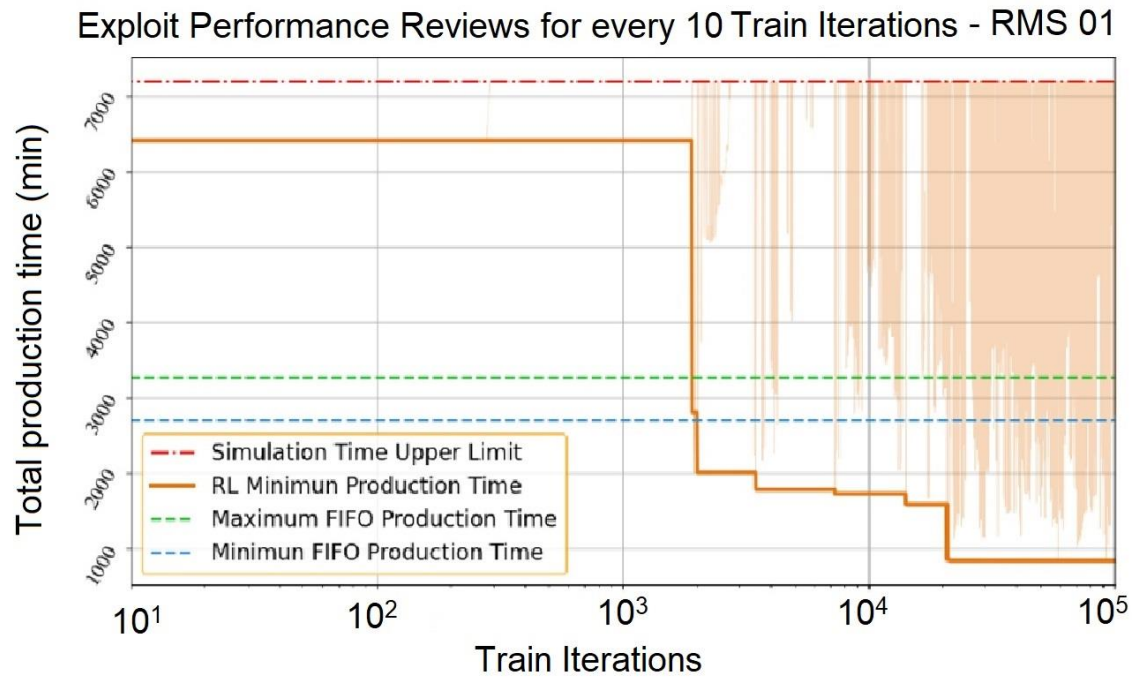


Figure 4-7 Detailed Training Performance Improvement

When an RMS perform in FIFO mode, raw material will dispatch to every free RMT only according to the order sequence, regardless of the RMT's current configuration. The proposed framework is designed 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. Figure 4-7 shows a detailed DQN agent training process on RMS01. Due to different initial RMT configurations, FIFO can finish the fixed order list in around 3000 mins. Starting from a poor performance level, the agent quickly outperforms FIFO after only training several thousand train iterations. The agent achieved its optimal strategy at around 20 thousand iterations.

4.1.2 Experiment 1-2: Advanced DQN Algorithm Validation

The basic DQN agent training process shows that the performance cannot converge at a steady phase. Therefore, aiming to improve the training stability, two Duelling Double DQN (DDDQN) agents are set for RMS02 and RMS03. Figure 4-8 shows that the DDDQN agent performs more stably after converging.

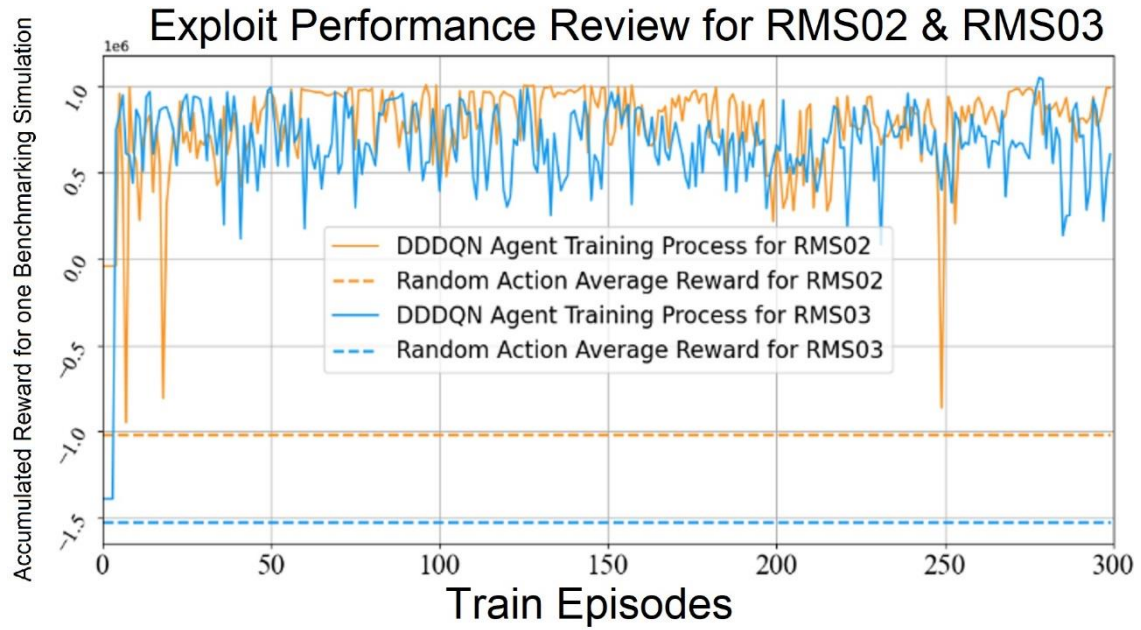


Figure 4-8 Advanced DQN Algorithms Training Process

4.1.3 Case Study One Summary

This case study suggests that DQN scheduling agents have great performance advantage potential. In very limited simulation turns, agents outperform a traditional dispatch rule, first-in-first-out. The training results show that all kinds of DQN agents can quickly converge to an above-average level. In deterministic testing, all agents found the optimal scheduling solution at a certain benchmark point.

Dueling Double DQN agents show advanced stability, as their inventor suggested. However, even if the converging capability and stability of DDDQN are much premium compared to the basic DQN, it is clear that an agent can perform way from optimal occasionally, regardless of the increasing training episodes. Therefore, keeping on optimising the agent stability with advanced techniques is necessary.

All agents converged in several episodes suggesting that the over-simplified RMS with a fixed order list cannot fully unveil the algorithm potential. The only randomness inside the environment is the initial configuration of every RMT.

The next case study tested the prioritised experience replay proposed in the framework. The RMS-DES also became much more complex to test the agent limitation.

4.2 Case Two: A Grid-world RMS

Case study two [128] presented in this section aims to evaluate the efficacy of a DQN agent in a complex system. Specifically, the agent is equipped with three advanced features, namely a double Q-network, dueling architecture, and prioritised experience replay (PER). In this case study, a stochastic order list generator is employed, although it is not dynamically updated. The capacity of the order list is held constant at a value identical to that used in the previous case study. However, unlike the previous case study, the jobs contained within the order list are no longer arranged in a fixed queue; rather, upon simulation initialization, the order generator assigns random product variants and quantities to the order list, with deadlines that are proportional to the requested quantity.

The RMS-DES followed the material handling system-focused approach proposed in the framework. As the environment is a multi-agent system, this case also studied a novel control method to bypass the exponentially growing action space issue. This control method also tried to help achieve an effect of multi-agent reinforcement learning in cooperation mode [66,129,130]. The idea behind this is central training with distributed execution.

To transfer the influence among AGVs, a special reward function is used for the following two experiences. In an RL agent training process, the interaction between the environment and the agent form transitions in a form like $\{s_t, a_t, r_t, s_{t+1}\}$. The r_t in a transition is related to one action a_t only. In this case study, r_t is separated into 2 parts. The local reward $^{AGV}r_t^{local}$, mostly dense reward, calculated according to the action itself. The global reward r_t^{global} is calculated according to some scenarios that have a global impact. The final reward store in transition is equal to $^{AGV}r_t^{local} + r_t^{global}$.

As the agent controls all AGVs, the action space for the AGVs is 6, which includes 4 kinds of movement: up, down, left, and right. The other two are waiting and interacting. Inside the RMS, AGVs are allowed to spend a little negative local reward to move in any direction until they hit a wall. If an AGV is

asked to hit a wall, this AGV will be forced to stay in front of the wall until the next turn and receive a big local negative reward. If the agent asks an AGV to stay at the current location and wait until the next event, this AGV will receive a small local negative reward from the RMS-DES. AGVs can also interact with RMTs and 3 kinds of warehouses, or inventory systems in the framework, across the RMS.

The first one is Module Storage (MDS). MDS provide certain kind of module that can be installed on RMTs. RMT can only manufacture raw materials with modules installed.

Raw material dispatch from the second kind of warehouse, Material Storage (MTS). AGVs can only deliver raw materials to RMT, install the correct module then trigger a manufacturing process event. After this process, the material in an RMT changes to a finished good.

An AGV now needs to transfer this product to the third kind of warehouse, a Finished Good Storage (FGS), where RMS check the stock level when an order is due. After every AGV interacts with RMS-DES, the RMS-DES will check whether an order needs to be delivered. If there is a due order and FSG has sufficient inventory, RMS will broadcast a huge global positive reward to all AGVs inside the system. On the other hand, if an FSG does not have enough stock while an order is due, all AGV agents will receive a big negative global reward. This is because the order that met the due date would be replaced by another randomly generated order regardless of whether it has been fulfilled or not.

Two experiments were conducted in this case study. Both case studies request the RMS to produce three kinds of products. Every product request one unique kind of module installed on one RMT. With a correct module, an RMT can keep taking the appointed kind of material and manufacturing product. The simulation time limitation is 28800 minutes.

Several key differences exist between the two case studies under consideration, including variations in the size of the grid-world, the number of

reconfigurable machine tools utilised, and the number of AGVs employed. These differences significantly impact the complexity and feasibility of various reinforcement learning approaches to scheduling and control in the respective manufacturing systems. In particular, the number of reconfigurable machine tools is a critical factor that affects the scheduling agent's ability to allocate resources and optimise production processes. A higher number of reconfigurable machine tools presents a more complex decision-making landscape that may require the agent to exhibit greater adaptability and responsiveness in its learning strategies. However, this increased complexity may also lead to longer training times and higher computational demands, which must be carefully managed to ensure the scalability and effectiveness of the reinforcement learning solution. Therefore, understanding the interplay between these factors and developing tailored approaches to address their specific challenges is paramount for realising the potential of reinforcement learning in diverse industrial settings. The differences show in the table below.

Table 4-4 Experiment 2-1 & 2-2 Settings

Experiment 2-1	Variables	Experiment 2-2
3	Product Variant	3
3	RMT Modules	3
28800 minutes	Simulation Time	28800 minutes
20×13	Grid-world Size	10×7
15	RMT	5
25	AGV	15

4.2.1 Experiment 2-1: An Anti-work Agent

The first experiment built a 20×13 grid-world RMS-DES, which installed 15 RMTs and 25 AGVs. The reconfiguring time from one module to another module is 5 minutes. The manufacturing time for one product is 3 minutes. The orders in the order list have a batch size from 50 to 1500, and the delivery time gap is a random value from 1000 to 5000 minutes. All RMTs and AGVs are initialised empty. The finished goods inventory was initialised with 500 stocks for all three products. The reward setting for every scenario mentioned in the last section is shown in Table 4-5.

Table 4-5 Experiment Case 2-1 Reward Function

Scenario #	Local Reward	Global Reward	Scenario #	Local Reward	Global Reward
1	-5	N/A	11	-1	N/A
2	-1,000	N/A	12	+10,000	N/A
3	-20	N/A	13	+10,000	N/A
4	+1,000	N/A	14	-3	N/A
5	-1	N/A	15	-1	N/A
6	-2	N/A	16	-3	N/A
7	+1000	N/A	17	-3	N/A
8	-1	N/A	18	N/A	+10k × order
9	-2	N/A	19	N/A	-1k × order
10	+10,000	+10k	20	-1	N/A

The reward quickly converges after only several rounds of training. However, the reward converged at an extremely low level, and there was no on-time delivery after the initial stock ran out.

To investigate the phenomenon, one AGV's moving trajectory in the first training episodes was sliced out and presented in Figure 4-9. Figure 4-9 is a 3D plot. Alongside the grid-world shape RMS, a third axis had been added to represent the simulation training iteration number. The trajectory clearly shows this AGV randomly wandering inside the system at the very beginning. However, with the greedy value reduced, the agent tends to ask AGVs to stay in the same place in

case gaining too many negative rewards. Considering this situation, the training process is cancelled though it should still converge if there is a computational power flood. A new reward mechanism was used for a new case study.

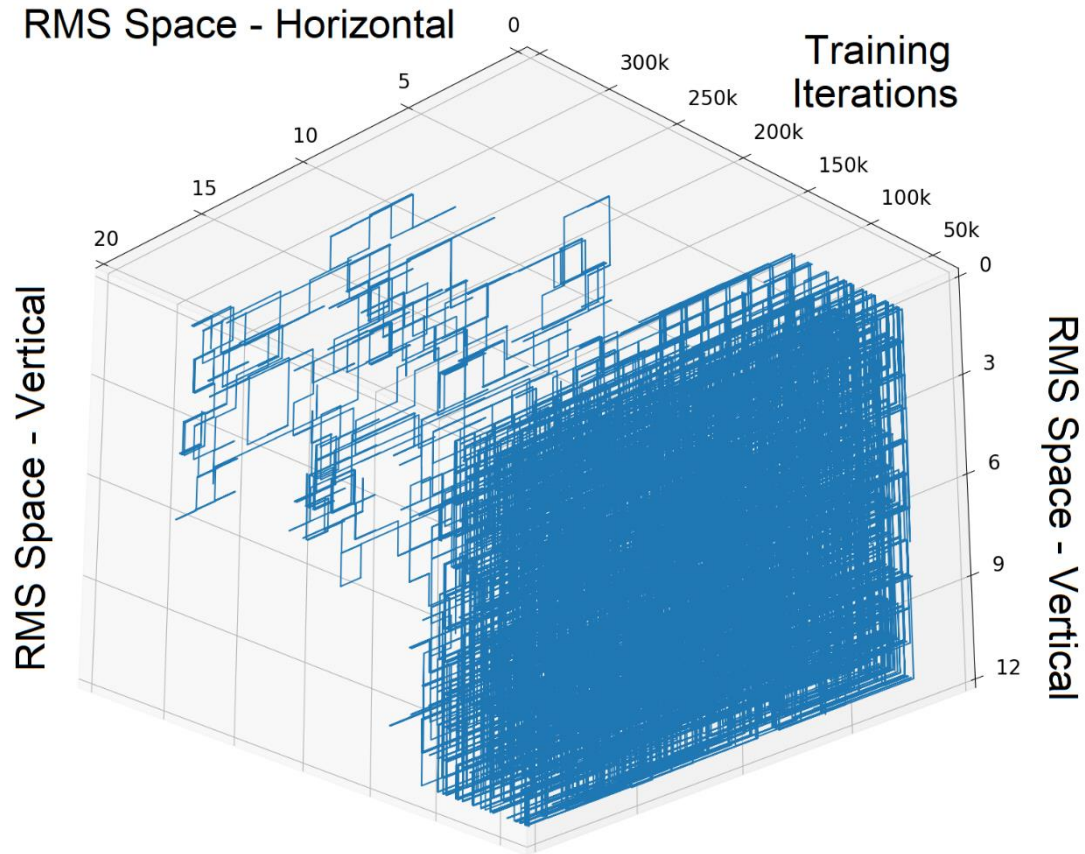


Figure 4-9 One AGV Movement Trajectory Sample

All AGVs' trajectories were reviewed manually. These trajectories share a similar pattern as Figure 4-9 suggest. This is not surprising as the agent is designed to control an individual AGV instead of manipulating them all simultaneously.

4.2.2 Experiment 2-2: Reward IS Critical

The agent in the first experiment showed a negative attitude toward exploring the environment. This first significantly reduces the chance that an agent found useful interaction. What makes it worse is that there are too many effective steps to be found and chained up to produce only one single product, while an order needs to be fulfilled with many products.

To further validate the effectiveness of the proposed framework, the current experiment needs to reduce the complexity of RMS-DES while adjusting the reward function to prevent the agent from becoming lazy.

In this experiment, the size of RMS was reduced to 10×7 , a quarter compared to the previous experiment. This RMS has 5 RMTs and 15 AGVs. The AGV number is exactly 3 times the number of RMTs. In an overly redundant example, one AGV will respond for transporting modules, one AGV will carry materials, and the last one will move the finished goods to the inventory.

Every time a simulation starts, two RMTs will be initialised empty, one RMT will be initialised having a finished good waiting to be carried away, one RMT will be initialised with a module wait for material, and the last RMT will be initialised with an on-going manufacturing process.

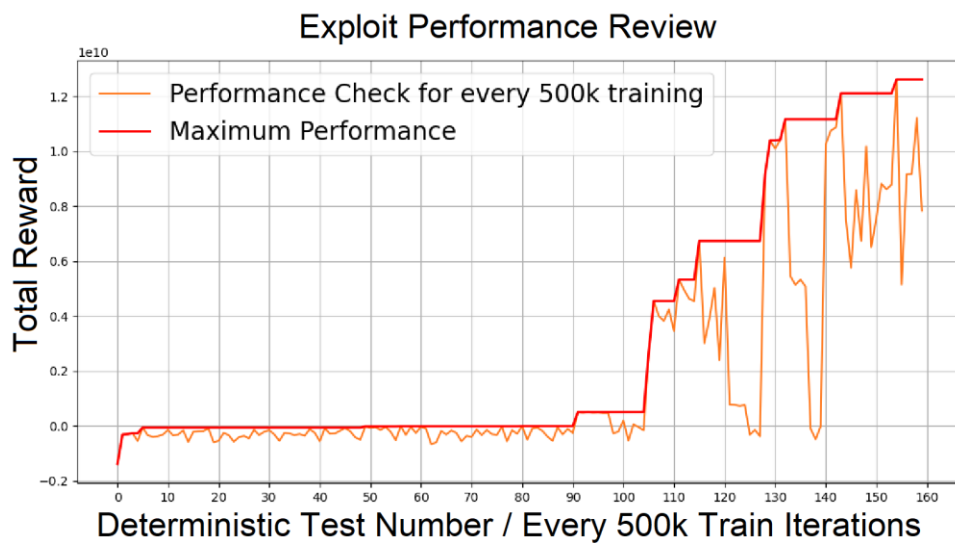
Every AGV is initialised with an empty cargo, a module, a material, or a finished good. These limitations and behaviour compositions are placed to help agents experience rewarding situations more frequently and constantly.

The new reward mechanism, shown in Table 4-6, increases the negative reward an AGV will receive if it chooses to keep at its current location. In addition, to increase the stimulation during the process, the local rewards generated by loading materials from material storage and delivering finished goods are increased.

Table 4-6 Experiment Case 2-2 Reward Setting

Scenario #	Local Reward	Global Reward	Scenario #	Local Reward	Global Reward
1	-1	N/A	11	-1	N/A
2	-1000	N/A	12	+1,000,000	N/A
3	-20	N/A	13	+10,000	N/A
4	+1k	N/A	14	-3	N/A
5	-1	N/A	15	-1	N/A
6	-2	N/A	16	-3	N/A
7	+100,000	N/A	17	-3	N/A
8	-1	N/A	18	N/A	+10k × order
9	-2	N/A	19	N/A	-1k × order
10	+1,000,000	+10k	20	-2	N/A

The PER capacity is set as 432,000, which equals transitions generated in one complete simulation. The mini-batch size for neural network training is 4096. Once the policy network gets trained 500 thousand times, the agent would use the deterministic policy, without implementing ε -greedy method, to test its performance. Training results are shown in Figures 4-10 and 4-11, presenting the reward-increasing curve and actual product an RMS could produce when testing.

**Figure 4-10 Experiment 2-2 Reward Trend**

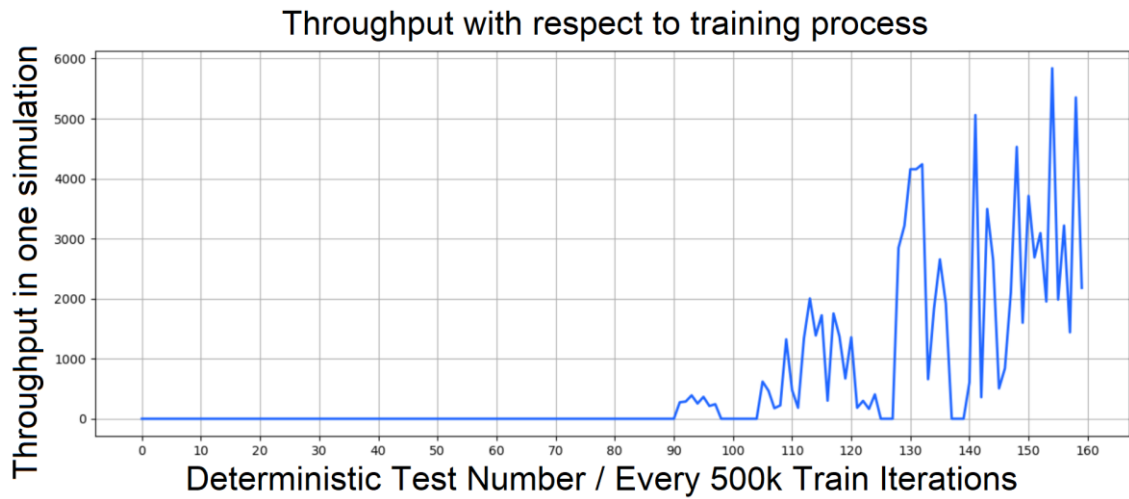


Figure 4-11 Productivity Trend

The training reward trend shows in Figure 4-10. It is clear that the agent quickly learned not to hit the wall. However, in the reward aspect, the training did not progress for a very long time until a sudden leap of around 45 million iterations. Figure 4-11 above shows the actual sum of products delivered from the RMS. Considering that the orders are randomly generated, the results look encouraging and present a good trend.

4.2.3 Case Study Two Summary

The present study aimed to investigate the effectiveness and applicability of a proposed training framework for solving medium and small-scale RMS scheduling problems. The results obtained from this case study suggest that the proposed training framework is not only applicable but also efficient in addressing such issues. This finding has significant implications for scheduling policy training, which is a critical area of research in operations management, as it provides an effective approach to address practical problems in real-world settings.

An essential aspect that emerged as a crucial learning outcome from this study was the significance of a training indicator. The process of training an agent for RL projects can be time-consuming, and it is not uncommon to invest several weeks or even months before the agent is proficiently trained. During this period, there may not be a clear reward growth trend that can be used to evaluate the agent's performance. In such scenarios, an indicator that can bypass the reward function is vital as it facilitates assessing whether the agent is exploring the environment adequately. The presence of a training indicator can help in avoiding potential issues that may arise from inadequate training and save time and resources in the long run.

Moreover, the proposed approach to scheduling policy training involved a carefully designed rewarding mechanism, imitation demonstrations, the combination of various behaviours, and a sufficient number of training episodes. These components work together to provide a comprehensive and robust training framework that can effectively tackle scheduling problems. The reward mechanism helps to incentivise the agent to perform desirable actions, while the imitation demonstrations provide a starting point for the agent to learn from. Additionally, combining various behaviours allows the agent to learn different strategies and approaches to solve scheduling problems.

Finally, the number of training episodes is a critical factor in the success of the proposed training framework. The training process involves the agent interacting with the environment and learning from its experiences. The more

training episodes that the agent undergoes, the more data it has to learn from, and the more refined its performance becomes. Therefore, by increasing the number of training episodes, the proposed scheduling policy training approach can still provide encouraging results, even in the absence of a clear reward growth trend.

In conclusion, the findings of this case study demonstrate that the proposed training framework is an effective and efficient approach to address scheduling problems in real-world settings. The inclusion of a training indicator, a carefully designed reward mechanism, imitation demonstrations, the combination of various behaviours, and a sufficient number of training episodes provides a comprehensive and robust training framework that can effectively train RL agents for scheduling policy problems.

4.3 Case Three: Machines Know How to Pull Properly

The findings from the second case study suggest that the utilisation of a material handling system-forced reconfigurable manufacturing system discrete event simulation model, which lacks a static layout, may not be the most optimal environment for training a scheduling agent from the ground up. In such a setting, the dynamic and unpredictable nature of the system can pose significant challenges for an RL scheduling agent to learn the underlying principles and strategies effectively. Moreover, the absence of a static layout in the model may result in a lack of structure and order, making it difficult for the scheduling agent to comprehend the system's behaviour accurately. As a result, it may be necessary to explore alternative methods for educating a scheduling agent in such contexts, such as incorporating more structure or introducing a more static layout in the simulation model. These findings have practical implications for industries that rely on material handling systems and reconfigurable manufacturing systems and can inform the development of effective training strategies for scheduling agents. This case study trains the schedule agent to use the RMT-focused approach proposed in the framework. The market demand simulator is the same as the one in the case study two, which is a stochastic order generator that sets the relevant model parameters, such as the order type and batch size. Thus, every single order is characterised by the product type (randomly setting one out of three) and the batch size (defined from a uniform random distribution ranging between 5 and 20 parts).

The deadline for completing an order is calculated based on the cycle time of the required product, the batch size, the available RMT number, and a random time variable with an expectation equal to reconfiguration time.

This assumption allows for the RMS in most orders to have just enough time to reconfigure RMTs, complete the production, and indirectly penalise the agent if it misses potential gains. There will always be 5 orders in the queue for the scheduling agent to plan ahead.

The RMS-DES in this case study is no longer a grid-world-like case study two but more similar to the one in case study one. The MHS is assumed to work

ideally in the just-in-time (JIT) mode [131], which means it can deliver the required material, module, or finished goods whenever the scheduling agent wants.

In this case study, the scheduling agent must manipulate every RMT to generate real-time schedules. These maintenance-free RMTs would equip a random module when initialising the RMS-DES. The same as case study two, 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.

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. Suppose the module that RMT is equipped with cannot follow the action order. In that case, the RMT will initiate the 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 have to spend some time reconfiguring 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. Figure 4-12 presents the flowchart below to formalise the procedure to enhance applicability and transferability.

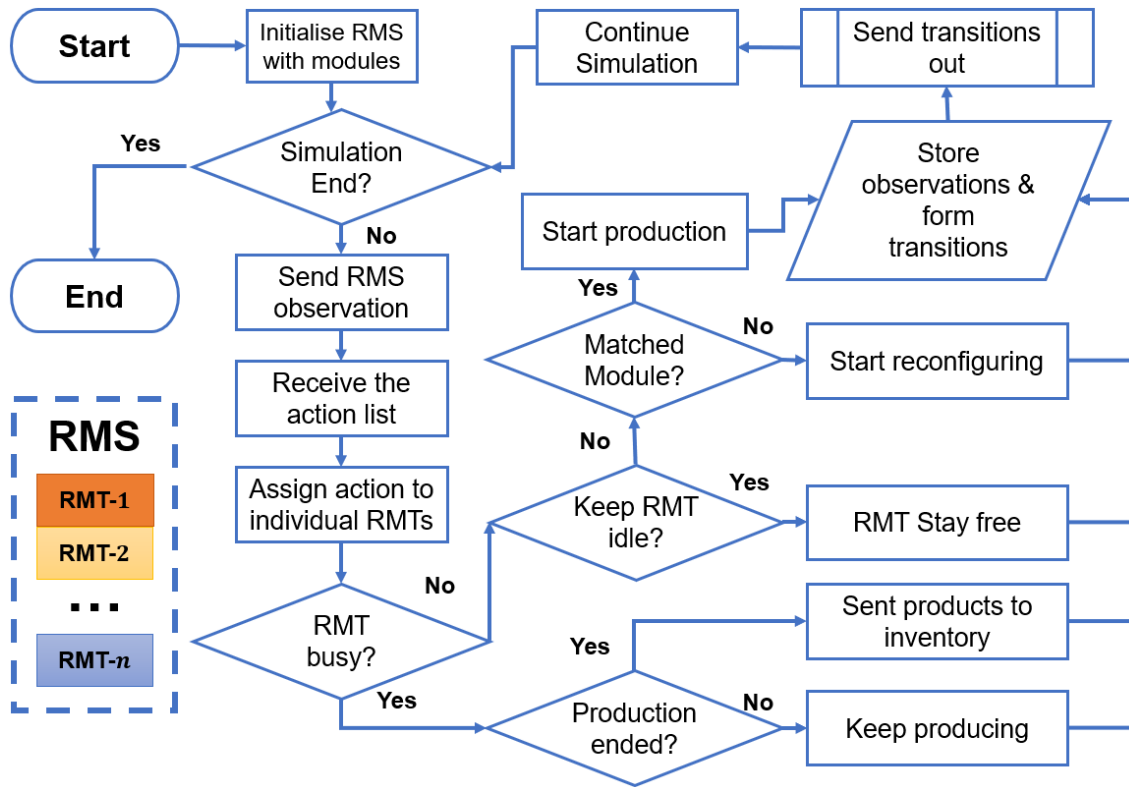


Figure 4-12 RMT-focused RMS Simulation Flowchart

As the general performance signal for the RL agent, the reward function of this case study is set as shown in Table 4-7 below.

Table 4-7 Case Study 3 Reward Function

Scenario	Reward Category	Detailed reward per timestep
Gain a product	Small positive	5
Keep a product inventory	Small negative	-1
Fulfil an order	Huge positive	$50 \times \text{Required quantity}$
Miss an order	Medium negative	$1 \times \text{Required quantity}$
Reconfigure an RMT	None	0
Keep an RMT rest	None	0

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

products remain in the inventory after the second order is delivered, this sum adds up to 8.

4.3.1 Experiment with a PPO Agent

The RL algorithm for this case is Proximal Policy Optimisation (PPO). The policy and value networks of the scheduling agent trained for this case study have a 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, respectively, before the 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 is set as 0.9996 so that the expected values of the first action can still be worth roughly up to 70% at the last step. The clip range for both networks is set as 0.1, and the maximum value for the gradient clipping is set as 0.5. The entropy coefficient set is 0.003 for forcing the agent to explore the unknown. When optimising the surrogate loss, the agent runs for 36 epochs.

Finally, the training process runs on a vectorised environment with 36 sub-environments. Figure 4-13 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 faded 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.

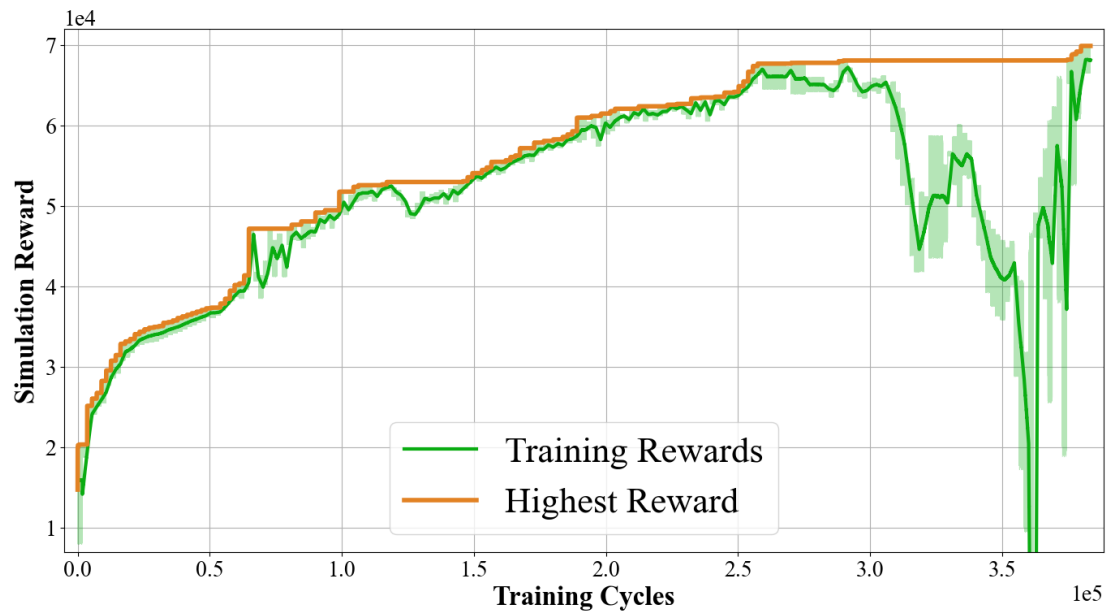


Figure 4-13 Exploit Performance Reward Trend

The reward trend of this case study is similar to other RL studies [75,91,132]. However, the big bounce before the end of this case study suggests that further review is needed.

As the major objective of this research, a scheduling agent should be able to achieve the highest possible order completion rate while keeping the inventory level low. In order to inspect this trade-off, the delivered orders versus the missed ones in one simulation would be compared first. Then, the inventory control performance would be inspected after that, and the RMT reconfiguration action would be inspected last.

Figure 4-14 presents three indices. The solid green line indicates how many orders the RMS can deliver once the agent is trained on average for 36 individual environments. The individual results are shown in the faded green area. Similar to the delivered order numbers, average and individual missed order numbers are shown in the solid red line and faded red area. Combining the data of delivered and missed orders, the average and individual order completion rate is plotted as the solid orange line and faded orange area. The order completion rate suggests a generally steady growing trend until it

achieves the theoretical limit. In such a condition, the major object of this project can be claimed to be achieved. However, Figure 4-14 alone cannot provide enough evidence to explain the reward jump and recovery in Figure 4-13.

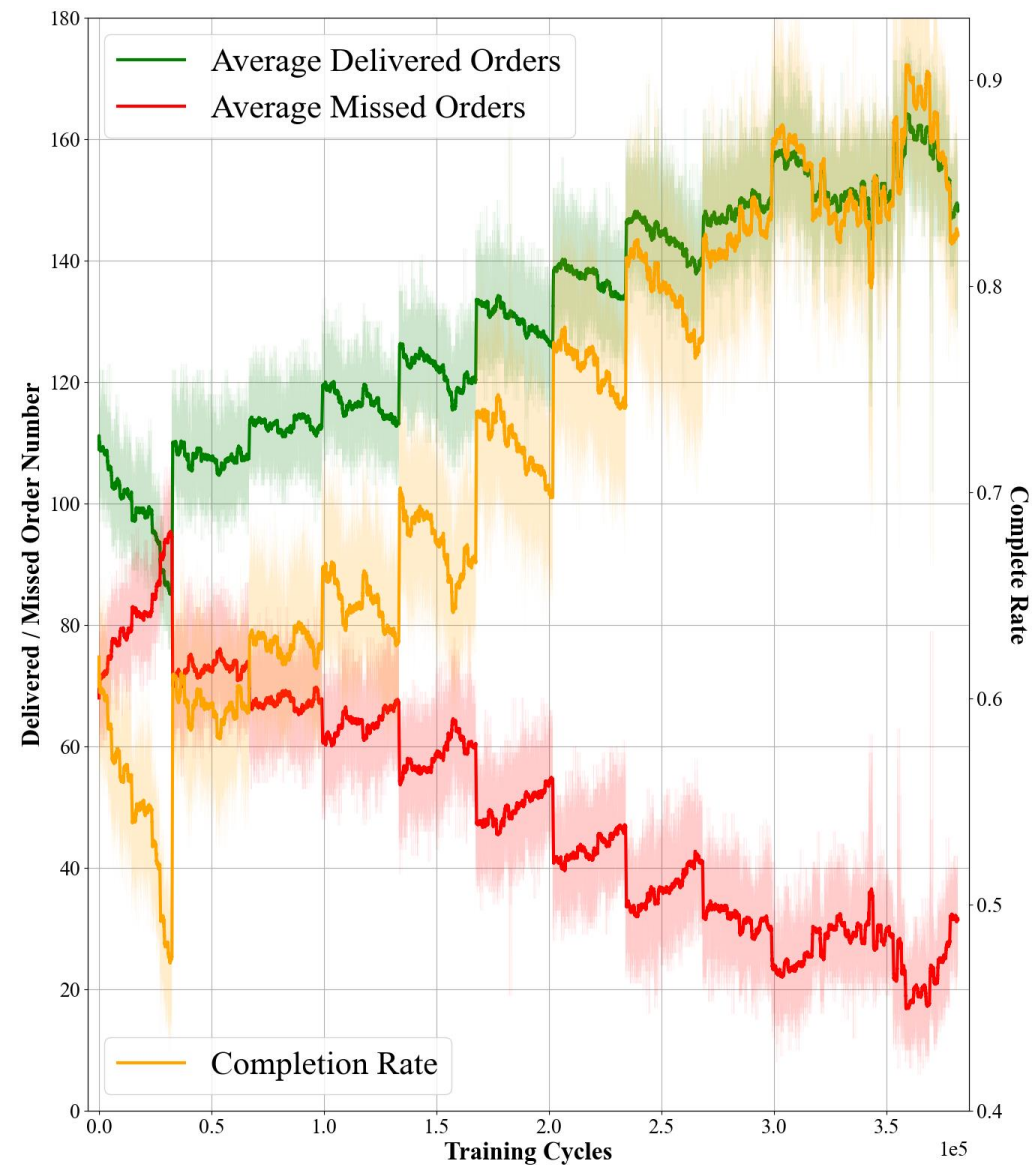


Figure 4-14 Order Completion Record

Figure 4-15 plots the cumulative inventory level as it changes with the training progress. The average cumulative inventory level is kept at around 3,000 before the reward drop significantly.

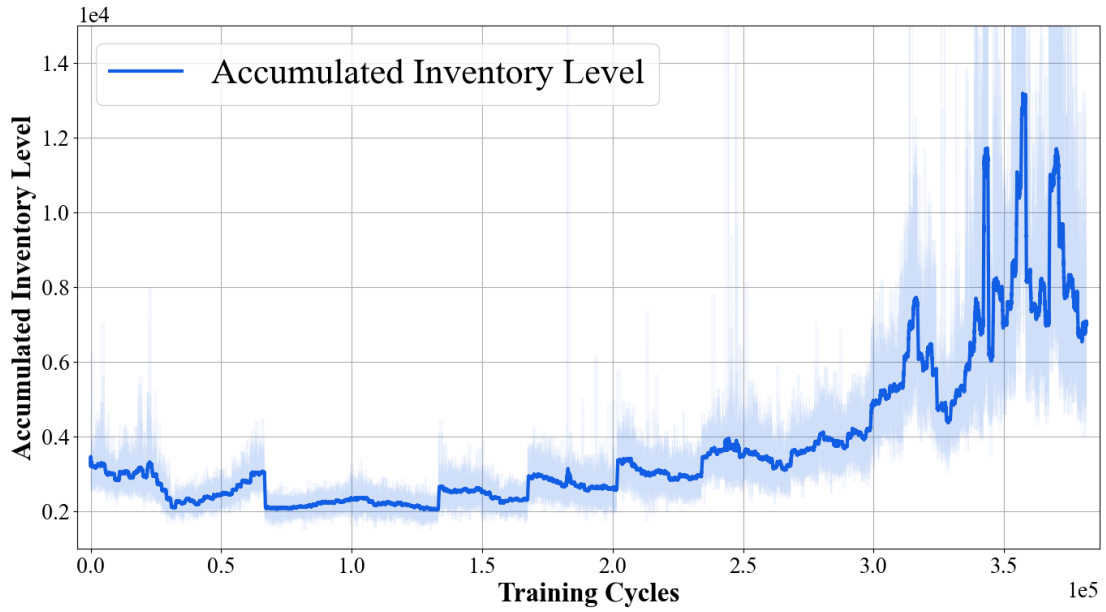


Figure 4-15 Accumulated Inventory Record

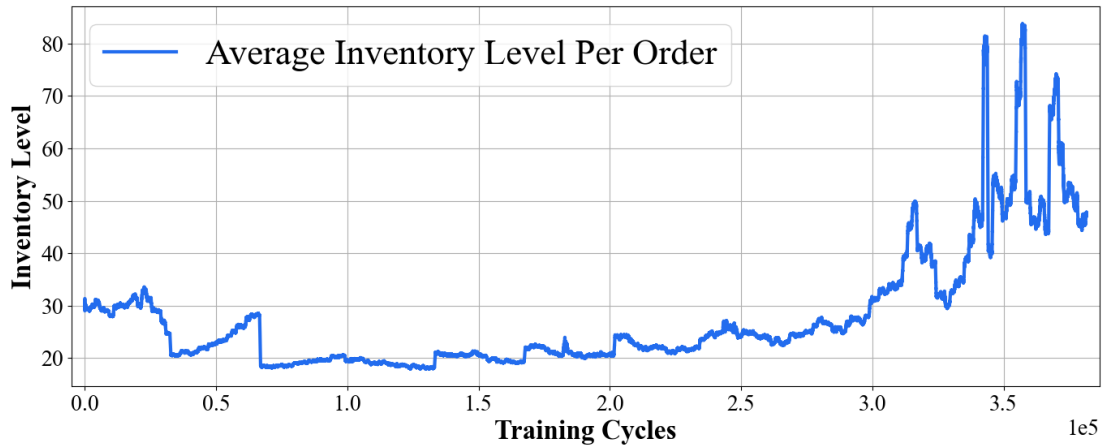


Figure 4-16 Average Inventory Level for Individual Orders

In Figure 4-16, the trend of the average inventory level is plotted and spread to individual orders by dividing the average accumulative inventory by the average of the delivered order number. Both trends in Figure 4-15 and 4-16 suggest that the agent tried to increase the stock in order to have a higher order completion rate as the instant reward is huge.

In Figure 4-17, 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 momentum, the agent has to seek another way to achieve a higher reward.

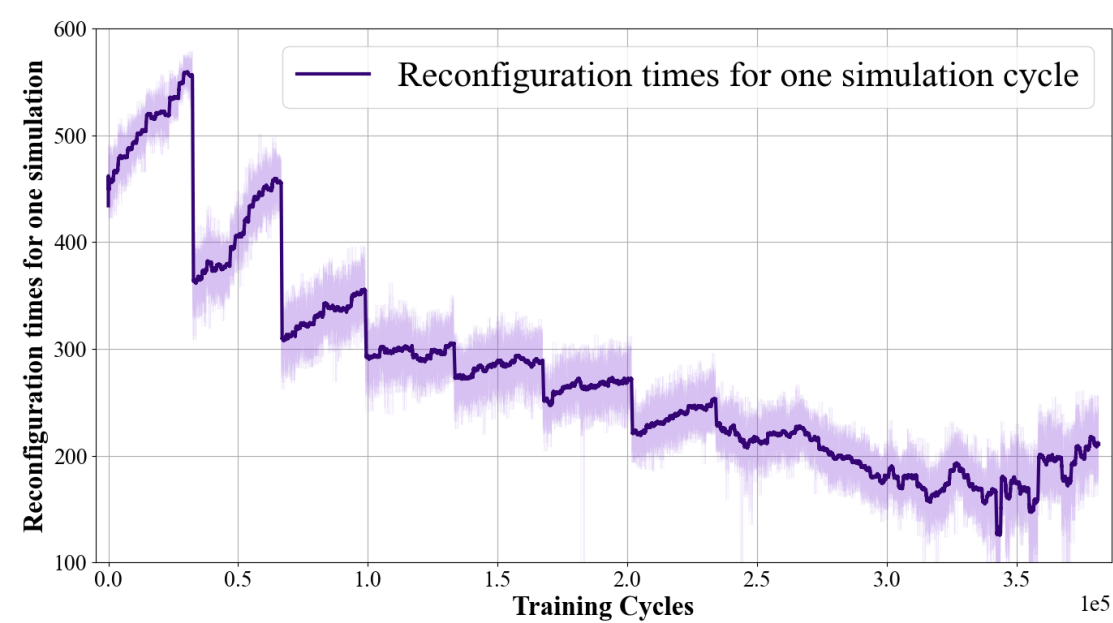


Figure 4-17 Total Reconfiguration Times Record

4.3.2 Case Study Three Summary

The PPO-based agent shows satisfactory resilience in fulfilling fluctuating demands and generates a smooth reward growth curve.

Same as the Case Study 2 suggest, emphasising the major optimising target can help the agent overcome some frustrating performance drop.

It also revealed that even if PPO was originally developed to avoid agents fall into some irreversible local optimal situations, like one policy update pushing the entire policy network into a region of the parameter space where the 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 not ideal hyperparameter setting.

The other major problem shown in the case study, similar to other RL projects, is the time-consuming training process. An RMS with good scalability 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.

In order to enhance the convergence speed and the oversimplified RMS, two supplementary experiments were performed, yielding noteworthy outcomes. The first experiment involved the implementation of an action mask (AM) with the intention of improving the training efficiency. Specifically, the AM functioned such that if an agent instructed an RMT to generate a product while the RMT was already occupied with production or reconfiguration, the command would be disregarded. Through the AM, the agent could only opt for the "stay rest / keep going" command until the RMT signalled its availability by raising an "I'm free now" event for the DES environment. Since an RMT should typically remain in the busy mode during the simulation, the AM was expected to assist the agent in filtering out useful trajectories. Nevertheless, the experimental findings were contrary to the anticipated results. Regardless of hyperparameter tuning, all PPO agents equipped with AM swiftly regressed into the "lazy mode" described in Case Study 2.

The other extra experiment makes the RMS-DES slightly more complex. When the prime products are stored in the inventory system, they can be retrieved as material again for further manufacturing. The market simulator also turned into a “hard mode” as an RMS can only fulfil at most 30% of orders on time. In such a setting, agents need only a few hours to learn to stock only one high-value-added product and its related material. The converging speed is roughly 100 times faster compared to an RMS which run in “lose-mode” and only need to produce prime products.

5 Discussion and Conclusion

This chapter will first highlight key literature review findings, then a review and discussion of agent performance from the preceding three case studies will follow. Next, usefulness, applicability, and generalisability of the proposed framework will be discussed. This chapter concludes with a discussion of the major contributions of this research, along with its limitations, and possible future research directions.

5.1 Key take-outs, discussion, and generalisability

As a paradigm aiming on taking advantages from both conventional dedicated production lines and flexible manufacturing systems, there is not a lot of real-world applications for validating the effectiveness of RMS in last two decades [1,3,7,112]. However, that does not mean RMS is completely outside the realm of academia. Research around the six core characteristics, and small-scale validating applications, brings more confidence for both academia and industry on adapting such a paradigm on upgrading existing legacies and future automated production lines.

Discrete event simulation (DES) is a powerful tool for modelling and analysing complex systems that involve discrete changes in state. When it comes to RL-based manufacturing system scheduling optimisation projects, DES is particularly suitable as the environment in this research for several reasons. Firstly, DES models modelled complex manufacturing processes with high fidelity, capturing both the deterministic and stochastic elements of the system. This means that the environment, outside the scope this research covered, can simulate a wide range of scenarios, including production line failures, machine downtime, and unexpected material shortages, which can impact production schedules. Secondly, DES provided a large amount of data that can be used for RL algorithms to learn from. For example, DES models generated time series data on machine utilisation, job completion times, and other key performance indicators, which had been used to train RL agents to make better scheduling decisions. Finally, DES can be used to test and validate RL-based scheduling

policies before deploying them in the real manufacturing environment, reducing the risk of costly errors or disruptions. Overall, DES is an essential tool for manufacturing system scheduling optimization projects that seek to leverage the power of RL to improve efficiency, reduce costs, and increase productivity. Once a problem is formulated into a Markov decision process, which is a nature of DES, reinforcement learning (RL) should be considered as a solution. The literature review section found that existing research applied RL algorithms to solve scheduling optimisation problems in simplistic ways. The contemporary research efforts are not fully aware of the most important capability of RL algorithms is finding a near-optimal solution on long-term in a dynamic environment with large observation space. Modelling the manufacturing systems in flexible job-shop scheduling problem form with no stochastic event driver cannot fully reveal the RL agents' potential on dynamic environment.

At the same time, RL algorithms are developing at a very fast rate in recent years. Value-based algorithms, like deep Q-network, are slowly being replaced by newer policy-based algorithms which took the advantages from value-based algorithms, like proximal policy optimisation.

All these drawbacks become the important drivers of proposing an end-to-end complete scheduling policy training framework. This framework was then validated through applying it on three case studies, which revealed several interesting key findings.

The first finding of this research is the effectiveness of applying such framework to a future-orientated manufacturing system. Case study one suggested that the proposed approach can provide similar performance on a conventional scheduling problem. With a static input and a rule-based environment, the implementation of RL algorithms on schedule optimisation, or generation, can be quite straightforward.

The second finding is the highlighted interface convenience. A shared pattern that ran through the three case studies is the isolation between simulation model building and RL agent constructing. The information exchange between

these two sections must follow the form “state, action, reward, next-state”, no matter how complicated an RMS state or how an agent remember an experience. This kind of isolation greatly improved the reliability of the framework because the model and algorithm can be upgraded separately without affecting compatibility.

The third finding of this research is that presented a clear approach to build reward functions for a reinforcement learning based schedule agent. This approach has been developed to bridge the agent and a discrete event simulation of a reconfigurable manufacturing system, thereby enabling the agent to learn from experience and make informed decisions. This finding is of paramount importance as it has the potential to enhance the efficiency and effectiveness of scheduling in manufacturing systems, which can lead to improved productivity and profitability. The proposed approach is expected to provide valuable insights for future research in this area and has the potential to revolutionise the way manufacturing systems are scheduled and managed.. A poorly designed reward function can easily make a training progress challenging, [133], such as the first experiment of the second case study. Once the reward function can provide a distinguished target, there could be an order of magnitude difference for the speed of an agent on finding a near-optimal solution, like the second extended experiment in the third case study.

So far, the versatility of the proposed framework had been presented naturally. The next section will present the general contribution of knowledge of this research.

5.2 Key contributions to knowledge

The main contribution of this PhD research is the development of an RL-based end-to-end framework that can be employed to solve scheduling problems in any RMS context.

There are three major novelties of this research: firstly, this research addresses a research area, which was identified from the literature review, that has not been thoroughly studied before. Secondly, the integration of the modelling approaches developed in this research in a unified framework that embraces the core characteristics of RMS has not been performed before within the context of the aim of this research. The third novelty of this study shed new light on the effectiveness of agent training time in improving the performance of the scheduling process as this relationship has not been thoroughly inspected in previous RL based schedule optimisation research.

The literature review presented in Chapter 2 aims to bridge an existing research gap in the field and highlight the need for further investigation into the area under consideration. Despite the wealth of literature available on the topic, there remains a dearth of comprehensive and up-to-date research relevant to the objectives of this study. Additionally, this thesis introduces a novel approach to framework development by utilising a unique combination of methods that have not been previously applied in this context. This innovative methodology is designed to address the limitations of existing approaches and to enhance the overall effectiveness of the proposed framework. Furthermore, the experimental results from the case studies conducted in this research demonstrate the practical applicability and effectiveness of the proposed framework in the context of RMS scheduling. Notably, the outcomes of these experiments have not been previously reported in the field, emphasising the novelty of the proposed framework. Finally, this study sought to evaluate the efficacy of cutting-edge training techniques in the context of the proposed framework. The results suggest that certain techniques may not live up to their purported benefits, raising questions about their practical utility and pointing towards

avenues for future research. Several sub-contributions have come across during the course of this study, which include:

- Two discrete-event simulation-based modelling frameworks which retain the core characteristics of RMS behaviour, and can be applied to evaluate the performance of any RMS. Both models are designed to evaluate the RL agent performance within an RMS setting. The machine-focused approach provides further evaluation potential on investigating the performance of every individual component inside the system, while extending the boundaries of the control policy of an RL agent.
- The realisation that the newer on-policy algorithm, PPO, perform much better than the conventional value-based off-policy algorithms, DQN, in the context of RMS scheduling problems.
- Contributed to an enhanced understanding of the field of RL in an RMS setting. This has been achieved through the validation of some of the cutting-edge training techniques, and through benchmarking RL-based solutions with other scheduling solutions found in the literature.

The next section will present the limitation and potential future research.

5.3 Limitations and future research directions

Although the three case studies clearly demonstrated that the proposed framework could generate desirable scheduling policies, the limitations found during the framework validation should be highlighted as well.

The last key finding suggests that the reward function reshaping should be the focus when implementing the framework, as the simulation model building and agent construction can be “out-sourcing”. However, this also means the person responsible for writing the reward function should have sufficient cross-disciplinary understanding on both how reinforcement learning work and how to do “feature engineering” [134] on simulation models.

The excessive time-consuming and computationally intensive training can significantly limit the scope of application of the framework. The training iteration number required to train a functional agent increased from tens of thousands in the first case study to hundreds of millions in the third case study. Based on the findings of the third case study, it was observed that the stability and rate of convergence of the agent can be enhanced by employing a greater degree of parallelisation of the environment. This involves distributing the training process across multiple instances, with each instance simulating a different version of the environment and contributing to the overall learning process. Additionally, using a larger batch size for neural network training can also help to improve the performance of the agent by enabling it to learn from a greater number of experiences at once. By leveraging these strategies, it may be possible to achieve more efficient and effective reinforcement learning outcomes in various applications. This limitation side-by-side justifies the rationality of using thousands of GPUs on AI training. This agent training limitation also conflicts with the original intention of reconfigurable manufacturing system paradigm which is fast response to change. The future research must keep a closer eye on compute-efficient improvements [135].

Alongside the framework implementation problems, the experiments conducted throughout a number of cases also highlighted some issues of this research.

The demand fluctuation assumption is rather simplistic and not necessarily realistic in reflecting many real-life settings. The strict, or no, deadline assumption is an overly simplified one, which would typically result in inaccurate representation of the real system. Therefore, deadlines should be treated with a certain degree of tolerance or flexibility (e.g., similar to the soft constraints in mathematical programming), which would result in a more accurate reflection of the modelled system.

The simplified grid-world RMS-DES with AGV-centric control logic is the most complex experiment setting in this research. Such simulation model was developed at high-level, in order to improve the applicability to wide RMS settings, therefore applying the framework on a real case study, with more real-life complexities is necessary for further validation. Future work on more complex real-life case studies can also further integrate and expand cooperative behaviours among agents.

This research focused on improving the convertibility of an RMS by training a smart scheduling policy. However, the numbers of RMTs in all three case studies are fixed, which means current setup lacks consideration of scalability and diagnosability, such as breakdowns, nor scalability, adding or removing machines or modules from systems, of an RMS. The pursuit of all possible enhancements aimed at the six core characteristics of an RMS poses a challenge for the reinforcement learning-based scheduling agents, particularly with respect to their observation space and action space. The dynamic nature of the manufacturing environment, in which factors such as machine availability, job specifications, and resource constraints are subject to frequent change, necessitates a high degree of adaptability and responsiveness on the part of the scheduling agent. To this end, reinforcement learning has emerged as a promising approach for enabling autonomous decision-making in such complex and dynamic systems. However, the ability of the agent to learn and generalise across a wide range of scenarios is contingent upon the availability of a sufficiently diverse and comprehensive observation space and action space. Therefore, the development of effective techniques for expanding and

optimising these spaces is crucial for the successful deployment of reinforcement learning-based scheduling agents in RMS settings. These changes will challenge the scheduling agent on transfer learning [136] and practical deployment.

5.4 Final remarks about potential industrial application

As the concept “simulation is the heart of digital twins” becomes a growing consensus [137], the mechanism of complex manufacturing system can be transferred into system simulation models easier than ever before [138,139]. This trend suggests operators could spend less time to check how individual components of a manufacturing system work and how these components influent each other. This does not mean the in-depth understanding is not important anymore, because data augmentation [140] would become a critical skill to balance data from the physical counterpart directly or deduct from the digital twin simulation.

Another promising trend is that reliable reinforcement learning libraries or implementations, like Stable-Baselines3 [141], are developing rapidly. The parallel training structure presented in previous section suggests the computational power bottleneck for small validation projects can be solved by the implementing of cloud computing.

Both simulation environment technologies and reinforcement learning algorithms are developing on optimistic expectations. These two trends emphasise the challenge that system engineers and field operators need to transform or extending current operation research knowledge into explicit reward shaping knowledge. The kind of work need significant amount of investment on time and human intelligence.

While considering the systems or models as black boxes, feature engineering techniques, a fundamental topic in machine learning, becomes a critical issue that should not be overlooked. A manufacturing system in production with its digital twins streams huge amounts of data in real time. It is certainly a challenging work to transform these numbers into information, or namely observation in RL context, for both artificial agents and humans to understand.

REFERENCES

1. Koren Y., Gu X., Guo W. Reconfigurable manufacturing systems: Principles, design, and future trends. *Frontiers of Mechanical Engineering*. 2018; 13(2): 121–136. Available at: DOI:10.1007/s11465-018-0483-0
2. El-Khalil R., Darwish Z. Flexible manufacturing systems performance in U.S. automotive manufacturing plants: a case study. *Production Planning and Control*. Taylor & Francis; 2019; 30(1): 48–59. Available at: DOI:10.1080/09537287.2018.1520318
3. Koren Y., Ulsoy AG. Reconfigurable manufacturing systems, engineering research center for reconfigurable machining systems (ERC/RMS) report# 1, the university of michigan. Ann Arbor. 1997;
4. Papakostas N., Georgoulas K., Koukas S. A novel platform for designing and evaluating Dynamic Manufacturing Networks. *CIRP Annals*. 2013; 62(1): 495–498. Available at: DOI:https://doi.org/10.1016/j.cirp.2013.03.111
5. Mourtzis D., Papakostas N., Mavrikios D., Makris S., Alexopoulos K. The role of simulation in digital manufacturing: applications and outlook. *International Journal of Computer Integrated Manufacturing*. Taylor & Francis; 2015; 28(1): 3–24. Available at: DOI:10.1080/0951192X.2013.800234
6. Papakostas N., Georgoulas K., Koukas S., Chryssolouris G. Organisation and operation of dynamic manufacturing networks. *International Journal of Computer Integrated Manufacturing*. Taylor & Francis; 2015; 28(8): 893–901. Available at: DOI:10.1080/0951192X.2014.933488
7. Koren Y., Heisel U., Jovane F., Moriwaki T., Pritschow G., Ulsoy G., et al. Reconfigurable Manufacturing Systems. *CIRP Annals*. 1999; 48(2): 527–540. Available at: DOI:10.1016/S0007-8506(07)63232-6

8. Abbasi M., Houshmand M. Production planning and performance optimization of reconfigurable manufacturing systems using genetic algorithm. *The International Journal of Advanced Manufacturing Technology*. Springer; 2011; 54(1): 373–392.
9. Napoleone A., Pozzetti A., Macchi M. Core Characteristics of Reconfigurability and their Influencing Elements. *IFAC-PapersOnLine*. 2018; 51(11): 116–121. Available at: DOI:10.1016/j.ifacol.2018.08.244
10. Akyol DE., Bayhan GM. A review on evolution of production scheduling with neural networks. *Computers & Industrial Engineering*. Elsevier; 2007; 53(1): 95–122.
11. Johnson SM. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*. Wiley Online Library; 1954; 1(1): 61–68.
12. Tang J., Emmanouilidis C., Salonitis K. Reconfigurable Manufacturing Systems Characteristics in Digital Twin Context. *IFAC-PapersOnLine*. 2020; 53(2): 10585–10590. Available at: DOI:https://doi.org/10.1016/j.ifacol.2020.12.2810
13. Son S-Y. Design principles and methodologies for reconfigurable machining systems. University of Michigan; 2000.
14. Spicer P., Carlo HJ. Integrating Reconfiguration Cost Into the Design of Multi-Period Scalable Reconfigurable Manufacturing Systems. *Journal of Manufacturing Science and Engineering*. 1 February 2007; 129(1): 202–210. Available at: DOI:10.1115/1.2383196
15. Aguilar A., Roman-Flores A., Huegel JC. Design, refinement, implementation and prototype testing of a reconfigurable lathe-mill. *Journal of manufacturing systems*. Elsevier; 2013; 32(2): 364–371.
16. Yigit AS., Ulsoy AG., Allahverdi A. Optimizing modular product design for reconfigurable manufacturing. *Journal of Intelligent Manufacturing*. Springer; 2002; 13(4): 309–316.

17. Yigit AS., Allahverdi A. Optimal selection of module instances for modular products in reconfigurable manufacturing systems. *International Journal of Production Research*. Taylor & Francis; 2003; 41(17): 4063–4074.
18. Eguia I., Molina JC., Lozano S., Racero J. Cell design and multi-period machine loading in cellular reconfigurable manufacturing systems with alternative routing. *International Journal of Production Research*. Taylor & Francis; 2017; 55(10): 2775–2790.
19. Bensmaine A., Dahane M., Benyoucef L. A new heuristic for integrated process planning and scheduling in reconfigurable manufacturing systems. *International Journal of Production Research*. Taylor & Francis; 2014; 52(12): 3583–3594.
20. Dou J., Su C., Zhao X. Mixed integer programming models for concurrent configuration design and scheduling in a reconfigurable manufacturing system. *Concurrent Engineering*. SAGE Publications Sage UK: London, England; 2020; 28(1): 32–46.
21. Dou J., Zhao X., Sun Q. Robust Optimization Models of Integrated Configuration Design and Scheduling for Reconfigurable Flowline. 2019 IEEE International Conference on Mechatronics and Automation (ICMA). 2019. pp. 70–75.
22. Abbasi M., Houshmand M., others. Production planning of reconfigurable manufacturing systems with stochastic demands using Tabu search. *International Journal of Manufacturing Technology and Management*. 2009; 17(1): 125.
23. Eguia I., Racero J., Guerrero F., Lozano S. Cell formation and scheduling of part families for reconfigurable cellular manufacturing systems using Tabu search. *Simulation*. Sage Publications Sage UK: London, England; 2013; 89(9): 1056–1072.
24. Musharavati F., Hamouda ASM. Enhanced simulated-annealing-based algorithms and their applications to process planning in reconfigurable

- manufacturing systems. *Advances in Engineering Software*. Elsevier; 2012; 45(1): 80–90.
25. Musharavati F., Hamouda AMS. Simulated annealing with auxiliary knowledge for process planning optimization in reconfigurable manufacturing. *Robotics and Computer-Integrated Manufacturing*. Elsevier; 2012; 28(2): 113–131.
 26. Haddou Benderbal H., Dahane M., Benyoucef L. Modularity assessment in reconfigurable manufacturing system (RMS) design: an Archived Multi-Objective Simulated Annealing-based approach. *The International Journal of Advanced Manufacturing Technology*. Springer; 2018; 94(1): 729–749.
 27. Borisovsky PA., Delorme X., Dolgui A. Genetic algorithm for balancing reconfigurable machining lines. *Computers & Industrial Engineering*. Elsevier; 2013; 66(3): 541–547.
 28. Bryan A., Hu SJ., Koren Y. Assembly system reconfiguration planning using genetic algorithm. *Engineering Systems Design and Analysis*. 2008. pp. 163–171.
 29. Dou J., Zhao X., Su C. An improved genetic algorithm for optimization of operation sequencing. 2018 IEEE International Conference on Mechatronics and Automation (ICMA). 2018. pp. 695–700.
 30. Li J., Wang A., Tang C. Production planning in virtual cell of reconfiguration manufacturing system using genetic algorithm. *The International Journal of Advanced Manufacturing Technology*. Springer; 2014; 74(1): 47–64.
 31. Ye H., Liang M. Simultaneous modular product scheduling and manufacturing cell reconfiguration using a genetic algorithm. 2006;
 32. Li A., Xie N. A Robust Scheduling for Reconfigurable Manufacturing System Using Petri Nets and Genetic Algorithm. 2006 6th World Congress on Intelligent Control and Automation. 2006. pp. 7302–7306. Available at: DOI:10.1109/WCICA.2006.1714504

33. Bensmaine A., Dahane M., Benyoucef L. A non-dominated sorting genetic algorithm based approach for optimal machines selection in reconfigurable manufacturing environment. *Computers & Industrial Engineering*. Elsevier; 2013; 66(3): 519–524.
34. Chaube A., Benyoucef L., Tiwari MK. An adapted NSGA-2 algorithm based dynamic process plan generation for a reconfigurable manufacturing system. *Journal of Intelligent Manufacturing*. Springer; 2012; 23(4): 1141–1155.
35. Dou J., Li J., Su C. Bi-objective optimization of integrating configuration generation and scheduling for reconfigurable flow lines using NSGA-II. *The International Journal of Advanced Manufacturing Technology*. Springer; 2016; 86(5): 1945–1962.
36. Goyal KK., Jain PK., Jain M. Optimal configuration selection for reconfigurable manufacturing system using NSGA II and TOPSIS. *International Journal of Production Research*. Taylor & Francis; 2012; 50(15): 4175–4191.
37. Spicer P., Carlo HJ. Integrating reconfiguration cost into the design of multi-period scalable reconfigurable manufacturing systems. 2007;
38. Bryan A., Hu SJ., Koren Y. Assembly system reconfiguration planning. *Journal of Manufacturing Science and Engineering*. American Society of Mechanical Engineers Digital Collection; 2013; 135(4).
39. Galán R. Hybrid heuristic approaches for scheduling in reconfigurable manufacturing systems. *Metaheuristics for scheduling in industrial and manufacturing applications*. Springer; 2008. pp. 211–253.
40. Bryan A., Hu SJ., Koren Y. Assembly system reconfiguration planning using genetic algorithm. *Engineering Systems Design and Analysis*. 2008. pp. 163–171.
41. Abbasi M., Houshmand M., others. Production planning of reconfigurable manufacturing systems with stochastic demands using Tabu search.

International Journal of Manufacturing Technology and Management. 2009; 17(1): 125.

42. Abbasi M., Houshmand M. Production planning and performance optimization of reconfigurable manufacturing systems using genetic algorithm. The International Journal of Advanced Manufacturing Technology. 2011; 54(1): 373–392. Available at: DOI:10.1007/s00170-010-2914-x
43. Carlo HJ., Patrick Spicer J., Rivera-Silva A. Simultaneous consideration of scalable-reconfigurable manufacturing system investment and operating costs. Journal of manufacturing science and engineering. American Society of Mechanical Engineers Digital Collection; 2012; 134(1).
44. Nehzati T., Ismail N., Aziz FA., Hosseini SA. Research outline on reconfigurable manufacturing system production scheduling employing fuzzy logic. International Journal of Information and Electronics Engineering. IACSIT Press; 2012; 2(5): 813–816.
45. Borisovsky PA., Delorme X., Dolgui A. Balancing reconfigurable machining lines via a set partitioning model. International Journal of Production Research. Taylor & Francis; 2014; 52(13): 4026–4036.
46. Bensmaine A., Dahane M., Benyoucef L. Process plan generation in reconfigurable manufacturing systems using AMOSA and TOPSIS. IFAC Proceedings Volumes. Elsevier; 2012; 45(6): 560–565.
47. Yu J-M., Doh H-H., Kim J-S., Kwon Y-J., Lee D-H., Nam S-H. Input sequencing and scheduling for a reconfigurable manufacturing system with a limited number of fixtures. The International Journal of Advanced Manufacturing Technology. Springer; 2013; 67(1): 157–169.
48. Choi Y-C., Xirouchakis P. A holistic production planning approach in a reconfigurable manufacturing system with energy consumption and

- environmental effects. *International Journal of Computer Integrated Manufacturing*. Taylor & Francis; 2015; 28(4): 379–394.
49. Rabbani M., Samavati M., Ziaee MS., Rafiei H. Reconfigurable dynamic cellular manufacturing system: a new bi-objective mathematical model. *RAIRO-Operations Research*. EDP Sciences; 2014; 48(1): 75–102.
 50. Moghaddam SK., Houshmand M., Fatahi Valilai O. Configuration design in scalable reconfigurable manufacturing systems (RMS); a case of single-product flow line (SPFL). *International Journal of Production Research*. Taylor & Francis; 2018; 56(11): 3932–3954.
 51. Moghaddam SK., Houshmand M., Saitou K., Fatahi Valilai O. Configuration design of scalable reconfigurable manufacturing systems for part family. *International Journal of Production Research*. Taylor & Francis; 2020; 58(10): 2974–2996.
 52. Ashraf M., Hasan F. Configuration selection for a reconfigurable manufacturing flow line involving part production with operation constraints. *The international journal of advanced manufacturing technology*. Springer; 2018; 98(5): 2137–2156.
 53. Prasad D., Jayswal SC. Reconfigurability consideration and scheduling of products in a manufacturing industry. *International Journal of Production Research*. Taylor & Francis; 2018; 56(19): 6430–6449.
 54. Ghanei S., AlGeddawy T. A new model for sustainable changeability and production planning. *Procedia CIRP*. Elsevier; 2016; 57: 522–526.
 55. Dou J., Li J., Xia D., Zhao X. A multi-objective particle swarm optimisation for integrated configuration design and scheduling in reconfigurable manufacturing system. *International Journal of Production Research*. Taylor & Francis; 2021; 59(13): 3975–3995.
 56. Haddou Benderbal H., Dahane M., Benyoucef L. Flexibility-based multi-objective approach for machines selection in reconfigurable manufacturing system (RMS) design under unavailability constraints.

- International Journal of Production Research. Taylor & Francis; 2017; 55(20): 6033–6051.
57. Asghar E., Baqai AA., Homri L., others. Optimum machine capabilities for reconfigurable manufacturing systems. The International Journal of Advanced Manufacturing Technology. Springer; 2018; 95(9): 4397–4417.
 58. Mahmoodjanloo M., Tavakkoli-Moghaddam R., Baboli A., Bozorgi-Amiri A. Flexible job shop scheduling problem with reconfigurable machine tools: An improved differential evolution algorithm. Applied Soft Computing. Elsevier; 2020; 94: 106416.
 59. Mahmoodjanloo M., Tavakkoli-Moghaddam R., Baboli A., Bozorgi-Amiri A. Distributed job-shop rescheduling problem considering reconfigurability of machines: a self-adaptive hybrid equilibrium optimiser. International Journal of Production Research. Taylor & Francis; 2022; 60(16): 4973–4994.
 60. Bortolini M., Ferrari E., Galizia FG., Regattieri A. An optimisation model for the dynamic management of cellular reconfigurable manufacturing systems under auxiliary module availability constraints. Journal of Manufacturing Systems. Elsevier; 2021; 58: 442–451.
 61. Mahmoodjanloo M., Tavakkoli-Moghaddam R., Baboli A., Bozorgi-Amiri A. Flexible job shop scheduling problem with reconfigurable machine tools: An improved differential evolution algorithm. Applied Soft Computing. Elsevier; 2020; 94: 106416.
 62. Vahedi-Nouri B., Tavakkoli-Moghaddam R., Hanzálek Z., Dolgui A. Workforce planning and production scheduling in a reconfigurable manufacturing system facing the COVID-19 pandemic. Journal of Manufacturing Systems. Elsevier; 2022; 63: 563–574.
 63. Tao F., Zhang H., Liu A., Nee AYC. Digital twin in industry: State-of-the-art. IEEE Transactions on industrial informatics. IEEE; 2018; 15(4): 2405–2415.

64. Xie J., Gao L., Peng K., Li X., Li H. Review on flexible job shop scheduling. *IET Collaborative Intelligent Manufacturing*. Wiley Online Library; 2019; 1(3): 67–77.
65. Usuga Cadavid JP., Lamouri S., Grabot B., Pellerin R., Fortin A. Machine learning applied in production planning and control: a state-of-the-art in the era of industry 4.0. *Journal of Intelligent Manufacturing*. Springer; 2020; 31(6): 1531–1558.
66. Vinyals O., Babuschkin I., Czarnecki WM., Mathieu M., Dudzik A., Chung J., et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*. Nature Publishing Group; 2019; 575(7782): 350–354.
67. Silver D., Hubert T., Schrittwieser J., Antonoglou I., Lai M., Guez A., et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*. American Association for the Advancement of Science; 2018; 362(6419): 1140–1144.
68. Silver D., Schrittwieser J., Simonyan K., Antonoglou I., Huang A., Guez A., et al. Mastering the game of go without human knowledge. *nature*. Nature Publishing Group; 2017; 550(7676): 354–359.
69. Silver D., Huang A., Maddison CJ., Guez A., Sifre L., van den Driessche G., et al. Mastering the game of Go with deep neural networks and tree search. *nature*. Nature Publishing Group; 2016; 529(7587): 484–489.
70. Lillicrap TP., Hunt JJ., Pritzel A., Heess N., Erez T., Tassa Y., et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*. 2015;
71. Zhang F., Leitner J., Milford M., Upcroft B., Corke P. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791*. 2015;
72. Ravuri S., Lenc K., Willson M., Kangin D., Lam R., Mirowski P., et al. Skilful precipitation nowcasting using deep generative models of radar.

- Nature. 2021; 597(7878): 672–677. Available at: DOI:10.1038/s41586-021-03854-z
73. Degraeve J., Felici F., Buchli J., Neunert M., Tracey B., Carpanese F., et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*. Nature Publishing Group; 2022; 602(7897): 414–419.
 74. Fawzi A., Balog M., Huang A., Hubert T., Romera-Paredes B., Barekatain M., et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*. Nature Publishing Group; 2022; 610(7930): 47–53.
 75. Sutton RS., Barto AG. Reinforcement Learning, second edition: An Introduction. MIT Press; 2018. Available at: <https://books.google.co.uk/books?id=uWV0DwAAQBAJ>
 76. Barto A., Duff M. Monte Carlo matrix inversion and reinforcement learning. *Advances in Neural Information Processing Systems*. 1993; 6.
 77. Bomberger EE. A dynamic programming approach to a lot size scheduling problem. *Management science*. INFORMS; 1966; 12(11): 778–784.
 78. Wirth C., Fürnkranz J. EPMC: Every visit preference Monte Carlo for reinforcement learning. *Asian Conference on Machine Learning*. 2013. pp. 483–497.
 79. Watkins CJCH., Dayan P. Q-learning. *Machine learning*. Springer; 1992; 8(3): 279–292.
 80. Bertsekas D., Tsitsiklis JN. *Neuro-dynamic programming*. Athena Scientific; 1996.
 81. Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*. 2013;

82. Mnih V., Kavukcuoglu K., Silver D., Rusu AA., Veness J., Bellemare MG., et al. Human-level control through deep reinforcement learning. *nature*. Nature Publishing Group; 2015; 518(7540): 529–533.
83. Hasselt H. Double Q-learning. *Advances in neural information processing systems*. 2010; 23.
84. van Hasselt H., Guez A., Silver D. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI conference on artificial intelligence*. 2016.
85. Schaul T., Quan J., Antonoglou I., Silver D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*. 2015;
86. Wang Z., Schaul T., Hessel M., Hasselt H., Lanctot M., Freitas N. Dueling network architectures for deep reinforcement learning. *International conference on machine learning*. 2016. pp. 1995–2003.
87. Barto AG., Sutton RS., Anderson CW. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*. IEEE; 1983; (5): 834–846.
88. Silver D., Hubert T., Schrittwieser J., Antonoglou I., Lai M., Guez A., et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*. 2017;
89. Schrittwieser J., Antonoglou I., Hubert T., Simonyan K., Sifre L., Schmitt S., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*. Nature Publishing Group; 2020; 588(7839): 604–609.
90. Mnih V., Badia AP., Mirza M., Graves A., Lillicrap T., Harley T., et al. Asynchronous methods for deep reinforcement learning. *International conference on machine learning*. 2016. pp. 1928–1937.
91. Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. 2017;

92. Zhang W., Dietterich T. High-performance job-shop scheduling with a time-delay TD (λ) network. *Advances in neural information processing systems*. 1995; 8.
93. Zhang W., Dietterich TG. A reinforcement learning approach to job-shop scheduling. *IJCAI*. 1995. pp. 1114–1120.
94. Zweben M., Davis E., Daun B., Deale M. Scheduling and rescheduling with iterative repair. 1992.
95. Riedmiller S., Riedmiller M. A neural reinforcement learning approach to learn local dispatching policies in production scheduling. *IJCAI*. 1999. pp. 764–771.
96. Paternina-Arboleda CD., Das TK. A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem. *Simulation Modelling Practice and Theory*. Elsevier; 2005; 13(5): 389–406.
97. Gabel T., Riedmiller M. Scaling adaptive agent-based reactive job-shop scheduling to large-scale problems. 2007 IEEE Symposium on Computational Intelligence in Scheduling. 2007. pp. 259–266.
98. Zhang Z., Zheng L., Weng MX. Dynamic parallel machine scheduling with mean weighted tardiness objective by Q-Learning. *The International Journal of Advanced Manufacturing Technology*. Springer; 2007; 34(9): 968–980.
99. Zhang Z., Zheng L., Li N., Wang W., Zhong S., Hu K. Minimizing mean weighted tardiness in unrelated parallel machine scheduling with reinforcement learning. *Computers & operations research*. Elsevier; 2012; 39(7): 1315–1324.
100. Palombarini J., Martinez E. SmartGantt–An intelligent system for real time rescheduling based on relational reinforcement learning. *Expert Systems with Applications*. Elsevier; 2012; 39(11): 10251–10268.

101. Stricker N., Kuhnle A., Sturm R., Friess S. Reinforcement learning for adaptive order dispatching in the semiconductor industry. *CIRP Annals*. Elsevier; 2018; 67(1): 511–514.
102. Waschneck B., Reichstaller A., Belzner L., Altenmüller T., Bauernhansl T., Knapp A., et al. Optimization of global production scheduling with deep reinforcement learning. *Procedia Cirp*. Elsevier; 2018; 72: 1264–1269.
103. Waschneck B., Reichstaller A., Belzner L., Altenmüller T., Bauernhansl T., Knapp A., et al. Deep reinforcement learning for semiconductor production scheduling. 2018 29th annual SEMI advanced semiconductor manufacturing conference (ASMC). 2018. pp. 301–306.
104. Waschneck B., Altenmüller T., Bauernhansl T., Kyek A. Production Scheduling in Complex Job Shops from an Industrie 4.0 Perspective: A Review and Challenges in the Semiconductor Industry. 2016.
105. Zhou L., Zhang L., Horn BKP. Deep reinforcement learning-based dynamic scheduling in smart manufacturing. *Procedia CIRP*. Elsevier B.V.; 2020. pp. 383–388. Available at: DOI:10.1016/j.procir.2020.05.163
106. Rummukainen H., Nurminen JK. Practical Reinforcement Learning - Experiences in Lot Scheduling Application **This work was in part supported by Business Finland, through project Engineering Rulez. *IFAC-PapersOnLine*. 2019; 52(13): 1415–1420. Available at: DOI:<https://doi.org/10.1016/j.ifacol.2019.11.397>
107. Kuhnle A., Röhrig N., Lanza G. Autonomous order dispatching in the semiconductor industry using reinforcement learning. *Procedia CIRP*. 2019; 79: 391–396. Available at: DOI:<https://doi.org/10.1016/j.procir.2019.02.101>
108. Zhang C., Song W., Cao Z., Zhang J., Tan PS., Chi X. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. In: Larochelle H, Ranzato M, Hadsell R, Balcan MF, Lin H (eds.) *Advances in Neural Information Processing Systems*. Curran Associates, Inc.; 2020.

- pp. 1621–1632. Available at: <https://proceedings.neurips.cc/paper/2020/file/11958dfce29b6709f48a9ba0387a2431-Paper.pdf>
109. Liu C-L., Chang C-C., Tseng C-J. Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems. IEEE Access. 2020; 8: 71752–71762. Available at: DOI:10.1109/ACCESS.2020.2987820
 110. Hubbs CD., Li C., Sahinidis N v., Grossmann IE., Wassick JM. A deep reinforcement learning approach for chemical production scheduling. Computers & Chemical Engineering. 2020; 141: 106982. Available at: DOI:<https://doi.org/10.1016/j.compchemeng.2020.106982>
 111. Chen X., Tian Y. Learning to Perform Local Rewriting for Combinatorial Optimization. In: Wallach H, Larochelle H, Beygelzimer A, d Alché-Buc F, Fox E, Garnett R (eds.) Advances in Neural Information Processing Systems. Curran Associates, Inc.; 2019. Available at: <https://proceedings.neurips.cc/paper/2019/file/131f383b434fdf48079bff1e44e2d9a5-Paper.pdf>
 112. Koren Y., Shpitalni M. Design of reconfigurable manufacturing systems. Journal of Manufacturing Systems. Elsevier Ltd; 2010; 29(4): 130–141. Available at: DOI:10.1016/j.jmsy.2011.01.001
 113. Koren Y., Heisel U., Jovane F., Moriwaki T., Pritschow G., Ulsoy G., et al. Reconfigurable Manufacturing Systems. CIRP Annals. Elsevier; 1 January 1999; 48(2): 527–540. Available at: DOI:10.1016/S0007-8506(07)63232-6 (Accessed: 1 February 2019)
 114. Siebers PO., Macal CM., Garnett J., Buxton D., Pidd M. Discrete-event simulation is dead, long live agent-based simulation! Journal of Simulation. 2010; 4(3): 204–210. Available at: DOI:10.1057/jos.2010.14
 115. Rabe M., Deininger M., Juan AA. Speeding up computational times in simheuristics combining genetic algorithms with discrete-Event

- simulation. *Simulation Modelling Practice and Theory*. 2020; 103(August 2019). Available at: DOI:10.1016/j.simpat.2020.102089
116. Rabe M., Dross F., Wuttke A. Combining a Discrete-event Simulation Model of a Logistics Network with Deep Reinforcement Learning. 2017; : 765–774.
 117. Khedri Liraviasl K., el Maraghy H., Hanafy M., Samy SN. A framework for modelling reconfigurable manufacturing systems using hybridized Discrete-Event and Agent-based simulation. *IFAC-PapersOnLine*. Elsevier Ltd.; 2015; 28(3): 1490–1495. Available at: DOI:10.1016/j.ifacol.2015.06.297
 118. Prajapat N., Tiwari A. A review of assembly optimisation applications using discrete event simulation. *International Journal of Computer Integrated Manufacturing*. 2017. pp. 215–228. Available at: DOI:10.1080/0951192X.2016.1145812
 119. Albakkoush S., Pagone E., Salonitis K. An approach to airline MRO operators planning and scheduling during aircraft line maintenance checks using discrete event simulation. *Procedia Manufacturing*. Elsevier B.V.; 2020; 54: 160–165. Available at: DOI:10.1016/j.promfg.2021.07.024
 120. Wang Z., Schaul T., Hessel M., Van Hasselt H., Lanctot M., De Frcitas N. Dueling Network Architectures for Deep Reinforcement Learning. 33rd International Conference on Machine Learning, ICML 2016. 2016; 4(9): 2939–2947.
 121. Zoph B., Le Q v. Neural Architecture Search with Reinforcement Learning. 4 November 2016; Available at: <http://arxiv.org/abs/1611.01578>
 122. Real E., Aggarwal A., Huang Y., Le Q v. Regularized Evolution for Image Classifier Architecture Search. 5 February 2018; Available at: <http://arxiv.org/abs/1802.01548>
 123. Real E., Moore S., Selle A., Saxena S., Suematsu YL., Tan J., et al. Large-Scale Evolution of Image Classifiers. 2017.

124. Zoph B., Vasudevan V., Shlens J., Le Q v. Learning Transferable Architectures for Scalable Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018.
125. Akiba T., Sano S., Yanase T., Ohta T., Koyama M. Optuna: A Next-generation Hyperparameter Optimization Framework. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Association for Computing Machinery; 2019. pp. 2623–2631. Available at: DOI:10.1145/3292500.3330701
126. Huang S., Ontañón S. A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. The International FLAIRS Conference Proceedings. University of Florida George A Smathers Libraries; May 2022; 35. Available at: DOI:10.32473/flairs.v35i.130584
127. Tang J., Salonitis K. A Deep Reinforcement Learning Based Scheduling Policy for Reconfigurable Manufacturing Systems. Procedia CIRP. Elsevier B.V.; 2021; 103: 1–7. Available at: DOI:10.1016/j.procir.2021.09.089
128. Tang J., Haddad Y., Salonitis K. Reconfigurable manufacturing system scheduling: a deep reinforcement learning approach. Procedia CIRP. 2022; 107: 1198–1203. Available at: DOI:https://doi.org/10.1016/j.procir.2022.05.131
129. OroojlooyJadid A., Hajinezhad D. A review of cooperative multi-agent deep reinforcement learning. arXiv. 2019;
130. Baer S., Bakakeu J., Meyes R., Meisen T. Multi-agent reinforcement learning for job shop scheduling in flexible manufacturing systems. Proceedings - 2019 2nd International Conference on Artificial Intelligence for Industries, AI4I 2019. 2019; (October): 22–25. Available at: DOI:10.1109/AI4I46381.2019.00014

131. Boysen N., Bock S. Scheduling just-in-time part supply for mixed-model assembly lines. *European Journal of Operational Research*. Elsevier B.V.; 2011; 211(1): 15–25. Available at: DOI:10.1016/j.ejor.2010.10.029
132. Schulman J., Levine S., Abbeel P., Jordan M., Moritz P. Trust Region Policy Optimization. In: Bach F, Blei D (eds.) *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France: PMLR; 2015. pp. 1889–1897. Available at: <https://proceedings.mlr.press/v37/schulman15.html>
133. Silver D., Singh S., Precup D., Sutton RS. Reward is enough. *Artificial Intelligence*. 2021; 299: 103535. Available at: DOI:<https://doi.org/10.1016/j.artint.2021.103535>
134. Lagoudakis MG., Parr R. Reinforcement Learning as Classification: Leveraging Modern Classifiers. 2003.
135. Humphreys PC., Guez A., Tieleman O., Sifre L., Weber T., Lillicrap T. Large-Scale Retrieval for Reinforcement Learning. *arXiv*; 2022. Available at: DOI:10.48550/ARXIV.2206.05314
136. Taylor ME., Stone P. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*. 2009.
137. Murphy R., Newell A., Hargaden V., Papakostas N. Machine learning technologies for order flowtime estimation in manufacturing systems. *Procedia CIRP*. 2019; 81: 701–706. Available at: DOI:<https://doi.org/10.1016/j.procir.2019.03.179>
138. Papakostas N., Newell A., Hargaden V. A novel paradigm for managing the product development process utilising blockchain technology principles. *CIRP Annals*. 2019; 68(1): 137–140. Available at: DOI:<https://doi.org/10.1016/j.cirp.2019.04.039>
139. Ramasubramanian AK., Connolly M., Mathew R., Papakostas N. Automatic simulation-based design and validation of robotic gripper

- fingers. *CIRP Annals*. 2022; 71(1): 137–140. Available at: DOI:<https://doi.org/10.1016/j.cirp.2022.04.054>
140. van Dyk DA., Meng X-L. The Art of Data Augmentation. *Journal of Computational and Graphical Statistics*. Taylor & Francis; 2001; 10(1): 1–50. Available at: DOI:10.1198/10618600152418584
141. Raffin A., Hill A., Gleave A., Kanervisto A., Ernestus M., Dormann N. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*. 2021. Available at: <https://github.com/DLR-RM/stable-baselines3>.

6 Appendices

6.1 Appendix One: The source code GitHub Repositories

https://github.com/Curtis-Tang/RMS_PhD_Thesis