Demystifying
Reinforcement Learning Approaches
for Production Scheduling

For attaining the academic degree of

Dr.-Ing.

from the Faculty of Mechanical Engineering
of the TU Dortmund University
Dissertation

submitted by

M.Sc. Alexandru Rînciog

from

Buzău, Romania

Dortmund, 2023

Oral examination date: 09. November 2023

Dean: Prof. Dr. Michael Henke

Reviewers:

Prof. Dr.-Ing. Anne Meyer (Karlsruher Institut für Technologie)
Jun.-Prof. Dr. Thomas Liebig (TU Dortmund University)

Funded by the German Research Foundation – project number 276879186 (GRK2193/2)

Abstract

Recent years has seen a sharp rise in interest pertaining to Reinforcement Learning (RL) approaches for production scheduling. This is because RL is seen as a an advantageous compromise between the two most typical scheduling solution approaches, namely priority rules and exact approaches. However, there are many variations of both production scheduling problems and RL solutions. Additionally, the RL production scheduling literature is characterized by a lack of standardization, which leads to the field being shrouded in mysticism. The burden of showcasing the exact situations where RL outshines other approaches still lies with the research community. To pave the way towards this goal, we make the following four contributions to the scientific community, aiding in the process of RL demystification. First, we develop a standardization framework for RL scheduling approaches using a comprehensive literature review as a conduit. Secondly, we design and implement FabricatioRL, an open-source benchmarking simulation framework for production scheduling covering a vast array of scheduling problems and ensuring experiment reproducibility. Thirdly, we create a set of baseline scheduling algorithms sharing some of the RL advantages. The set of RL-competitive algorithms consists of a Constraint Programming (CP) meta-heuristic developed by us, CP3, and two simulation-based approaches namely a novel approach we call Simulation Search and Monte Carlo Tree Search. Fourth and finally, we use FabricatioRL to build two benchmarking instances for two popular stochastic production scheduling problems, and run fully reproducible experiments on them, pitting Double Deep Q Networks (DDQN) and AlphaGo Zero (AZ) against the chosen baselines and priority rules. Our results show that AZ manages to marginally outperform priority rules and DDQN, but fails to outperform our competitive baselines.

Kurzfassung

In den letzten Jahren ist das Interesse an Ansätzen des Reinforcement Learning (RL) für Produktionsteuerung stark gestiegen. Dies liegt daran, dass RL als vorteilhafter Kompromiss zwischen den beiden typischsten Steuerungsansätzen, nämlich Prioritätsregeln und Neuplannung mittels exakten Verfahren, angesehen wird. Jedoch gibt es eine Vielzahl sowohl an Produktionsteuerungproblemen als auch an RL-Lösungesansätzen. Darüber hinaus ist die wissenschaftliche Literatur zu RL-Ansätzen für Produktionsteuerung von einem Mangel an Standardisierung gekennzeichnet, was dazu führt, dass das Feld in einer gewissen Mystik gehüllt ist. Die Last, die genauen Situationen aufzuzeigen, in denen RL andere Ansätze übertrifft, liegt immer noch bei der Forschungsgemeinschaft. Um den Weg zu diesem Ziel zu ebnen, leisten wir die folgenden vier Beiträge an die wissenschaftliche Gemeinschaft, die den Prozess der RL-Entmystifizierung unterstützen. Zunächst entwickeln wir ein Standardisierungs-Framework für RL-Produktionsteuerungsansätze über eine Systematische Literaturrecherche. Zweitens entwerfen und implementieren wir FabricatioRL, ein Open-Source Benchmarking-Simulations-Framework für die Produktionsteuerung, das eine Vielzahl an Produktionsteuerungsprobleme abdeckt. Insbesondere erleichtert das Simulations-Framework die Gewährleistung der Reproduzierbarkeit von Experimenten. Drittens wählen wir eine Menge an Baseline-Scheduling-Algorithmen, die die RL-Vorteile teilen. Der Satz an RL-kompetitiven Algorithmen besteht aus einer von uns entwickelten Constraint Programming (CP) Meta-Heuristik namens CP3, und zwei simulationsbasierten Ansätzen, nämlich ein von uns entwickelter neuartiger Ansatz namens Simulation Search, und Monte Carlo Tree Search. Nicht zuletzt wandeln wir zwei weit verbreitete Benchmarking-Instanzen für Produktionsplanung in Instanzen für Produktionsteuerung um und verwenden FabricatioRL um vollständig reproduzierbare Experimente hierauf durchzuführen. Dabei vergleichen wir zwei populäre RL-Ansätze, und zwar Double Deep Q Networks (DDQN) und AlphaGo Zero (AZ) mit den oben genannten Baselines, sowie mit einfachen Prioritätsregeln, hinsichtlich des erreichten Makespans. Unsere Ergebnisse zeigen, dass AZ es schafft, Prioritätsregeln und DDQN geringfügig zu übertreffen jedoch nicht die von uns vorgeschlagenen RL-kompetitiven Baselines.

Contents

List of Abbreviations **III**

List of Figures **V**

List of Tables **VII**

List of Equations **IX**

1 Introduction **1**
- 1.1 Problems in the RL Production Scheduling World . . . 2
- 1.2 Demystifying RL Production Scheduling Approaches . . . 5

2 Ordering Disorder:
A Standardization Framework for RL Production Scheduling **9**
- 2.1 Related Efforts . . . 11
- 2.2 Job Shop Scheduling Variants . . . 14
  - 2.2.1 Production Scheduling Problem Families . . . 15
  - 2.2.2 Additional Setup Complexity . . . 17
  - 2.2.3 Objectives . . . 23
- 2.3 RL Modeling for Job Shops . . . 26
  - 2.3.1 MDP Breakdown . . . 26
  - 2.3.2 States, Actions, Rewards . . . 33
  - 2.3.3 Agent Types . . . 38
- 2.4 Validation . . . 42
  - 2.4.1 Reproducibility . . . 42
  - 2.4.2 Evaluation . . . 47
- 2.5 Research Gaps . . . 50

3 FabricatioRL:
A Benchmarking Simulation Framework for RL Production Scheduling **53**
- 3.1 Requirements . . . 54
  - 3.1.1 Scheduling Setup . . . 55
  - 3.1.2 RL Modeling . . . 57
  - 3.1.3 Validation . . . 61
- 3.2 Implementation . . . 63
  - 3.2.1 Architecture . . . 63
  - 3.2.2 Initialization and Inputs . . . 64
  - 3.2.3 Interface Module . . . 69
  - 3.2.4 Core Module . . . 73
- 3.3 Visualization and Examples . . . 79
  - 3.3.1 Webapp and Logger . . . 80
  - 3.3.2 Usage Examples . . . 83

3.4 Concluding Remarks 84

4 From Theory to Implementation: Selected RL Scheduling Algorithms 87
4.1 Preliminaries 89
4.1.1 Markov Decision Process and the Generalized Policy Iteration . . . 89
4.1.2 Value-Based RL Approaches 92
4.1.3 Policy-Based RL Approaches 94
4.1.4 Actor-Critic RL Approaches 95
4.2 Selected RL Algorithms 95
4.2.1 Double Deep-Q Networks 96
4.2.2 AlphaGo Zero 99

5 Leveling the Playing Field: RL-Competitive Scheduling Baselines 105
5.1 Snapshot-Based Approaches 106
5.1.1 Simple Heuristics 106
5.1.2 Constraint Programming Heuristic 109
5.2 Simulation-Based Approaches 113
5.2.1 Simulation Search 113
5.2.2 Monte Carlo Tree Search 116
5.3 Remarks on Baseline RL-Competitiveness 120

6 Choosing through Experimentation: Setups, RL Design, Training and Evaluation 123
6.1 Scheduling Setup 124
6.1.1 Base Problems 124
6.1.2 Dynamic Problems 125
6.2 RL Design 130
6.2.1 Design Space Reduction 131
6.2.2 Reward Design 132
6.2.3 State Design 133
6.2.4 Action Design 140
6.3 Training and Evaluation 143
6.3.1 Experiment Setup 144
6.3.2 Model Selection 145
6.3.3 Evaluation 151

7 Conclusion and Future Work 159

References 181

Appendices I
Appendix A Literature Overview Tables III
Appendix B FabricatioRL Classes XV
Appendix C Feature Evolution Over Time XIX
Appendix D Further Optimization Goals Measured During Experiments . XXXIII

List of Abbreviations

RL	Reinforcement Learning	1
ML	Machine Learning	1
CP	Constraint Programming	2
EA	Evolutionary Algorithm	2
MILP	Mixed Integer Linear Programming	2
LPT	Longest Processing Time	2
AZ	AlphaGo Zero	3
DQN	Deep-Q Networks	3
GCN	Graph Convolutional Network	3
SOM	Self Organizing Maps	3
QL	Q-Learning	3
FCNN	Fully Connected Neural Network	3
API	Application Programming Interface	4
SimSearch	Simulation Search	5
DDQN	Dual Deep-Q Networks	5
MCTS	Monte Carlo Tree Search	5
MDP	Markov Decision Process	9
WoS	Web of Science Portal	9
OR	Operations Research	11
PPO	Proximal Policy Optimization	14
TRPO	Trust Region Policy Optimization	14
DAG	Directed Acyclic Graph	17
WIP	Work in Progress Window	25
FIFO	First In First Out	29
LIFO	Last In First Out	37
SPT	Shortest Processing Time	37
EDD	Earliest Due Date	37
DDDQN	Dueling Dual Deep-Q Networks	39
SARSA	State Action Reward State Action	39
DDPG	Deep Deterministic Policy Gradient	39
VAC	Value Actor Critic	39
A3C	Asynchronous Advantage Actor Critic	39
A2C	Advantage Actor Critic	39
NN	Neural Network	40

CNN Convolutional Neural Network 41
TDNN Time Delay Neural Network 41
RNN Recurrent Neural Network 41
RNG Random Number Generator 46
GenFJS Generalized Flexible Job Shop 56
JSON JavaScript Object Notation 80
KPI Key Performance Indicator 81
GPI Generalized Policy Iteration 89
MSE Mean Squared Error 93
PG Policy Gradient 94
SAT Boolean Satisfiability Problem 99
MES Manufacturing Execution System 106
LRPT Longest Remaining Processing Time 107
MOR Most Operations Remaining 107
MQO Most Queued Operations 107
SRPT Shortest Remaining Processing Time 107
LQO Least Queued Operations 107
LOR Least Operations Remaining 107
MTPO Most Time Per Operation 107
LQT Least Queued Time 107
LUDM Least Utilized Downstream Machine 107
MQT Most Queued Time 107
LTPO Least Time Per Operation 107
UCT Upper Confidence Bound for Trees 117
UCB1 Upper Confidence Bound 1 117
RELU Rectified Linear Unit 145
MAE Mean Absolute Error 147
VBS Virtual Best Selector 151

List of Figures

1.1 RL Publications . . . 3

2.1 Machine Setup Hierarchy . . . 16
2.2 Additional Constraints (β) . . . 18
2.3 Parallel Operations Examples . . . 21
2.4 Optimization Goals (γ) . . . 24
2.5 Basic RL Loop . . . 26
2.6 Reactive Production Scheduling Decisions . . . 28
2.7 MDP Breakdowns in Literature . . . 33
2.8 States in Literature . . . 36
2.9 Actions in Literature . . . 38
2.10 RL Classes in Literature . . . 40
2.11 RL Algorithms in Literature . . . 40
2.12 RL Functions in Literature . . . 41
2.13 RL Multi-Agents in Literature . . . 41
2.14 Setup Clarity in Literature . . . 44
2.15 Design Clarity in Literature . . . 45
2.16 Input Availability in Literature . . . 46
2.17 Simulation Availability in Literature . . . 46
2.18 Reproducible Stochasticity in Literature. . . . 47
2.19 Test-Train Split in Literature . . . 48
2.20 Baselining Sufficiency . . . 49
2.21 Cherry Picking Potential in RL Literature . . . 50

3.1 FabricatioRL Decisions . . . 59
3.2 FabricatioRL Class Diagram . . . 64
3.3 FabricatioRL Initialization . . . 65
3.4 Operation Duration Distributions . . . 67
3.5 DAG Generation . . . 68
3.6 FabricatioRL Interface Step . . . 72
3.7 FabricatioRL State Components . . . 74
3.8 FabricatioRL Trackers Components . . . 75
3.9 FabricatioRL Core Step . . . 79
3.10 Visualization Front-End . . . 81

4.1 Generalized Policy Iteration . . . 92

4.2 Doubele DQN 99
4.3 AZ MCTS 102
4.4 AZ Training 103

5.1 Simulation Search Algorithm 114
5.2 MCTS Algorithm 117

6.1 Benchmark Instances 125
6.2 Distribution of Operation Numbers in Literature. 126
6.3 WIP Size Selection 126
6.4 Analysis Data Acquisition with Simulation Search 129
6.5 FJc Load Behavior 129
6.6 Jm Load Behavior 130
6.7 Reward-Target Correlations 134
6.8 Masking Example 141
6.9 Heuristic Winners 143
6.10 Heuristic Ties 143
6.11 D1-DQN Learning Curves 147
6.12 D2-DQN Learning Curves 148
6.13 D3-DQN Learning Curves 149
6.14 D3-AZ Learning Curves 152
6.15 Scheduling Algorithm Runtimes 153
6.16 Results FJc 154
6.17 Results Jm 155

B.1 interface_templates and env_utils Module Classes. XV
B.2 interface, interface_input and interfece_RNG Module Classes. XVI
B.3 core, events, core_management and logger Module Classes. XVII
B.4 state Module Classes. XVIII

C.1 Jm Feature Behavior (1-8). XX
C.2 Jm Feature Behavior (9-16). XXI
C.3 Jm Feature Behavior (17-24). XXII
C.4 Jm Feature Behavior (25-32). XXIII
C.5 Jm Feature Behavior (33-40). XXIV
C.6 Jm Feature Behavior (41-48). XXV
C.7 FJc Feature Behavior (1-8). XXVI
C.8 FJc Feature Behavior (9-16). XXVII
C.9 FJc Feature Behavior (17-24). XXVIII
C.10 FJc Feature Behavior (25-32). XXIX
C.11 FJc Feature Behavior (33-40). XXX
C.12 FJc Feature Behavior (41-48). XXXI

List of Tables

2.1 MDP Decisions 32

3.1 Optimizer Configurations 70
3.2 Simulation Framework Requirements 85

4.1 Perceived RL Advantages 88
4.2 DDQN Parameters 98
4.3 AZ Parameters 104

5.1 Priority Rule Overview 108

6.1 Feature Overview 137
6.2 DQN Parameter Values 146
6.3 AZ Parameter Values 150
6.4 Makespan Result Overview 156

7.1 Attributes of Scheduling Solvers 160

A.1 Scheduling Setups in Literature. III
A.2 RL Designs in Literature. VIII
A.3 Validation of RL Experiments in Literature. XI

D.1 Flow Time Results. XXXIII
D.2 Tardiness Results. XXXIV
D.3 Utilization Results. XXXIV

List of Equations

4.1 MDP Dynamics Function 89
4.2 MDP State Transition Function 90
4.3 MDP Reward Function 90
4.4 MDP Markov Property 90
4.5 Discounted Future Reward 90
4.6 Action Value Function (1st Bellman Equation) 91
4.7 State Value Function (2nd Bellman Equation) 91
4.8 Optimal Action Value Function and Optimal Policy 91
4.9 Optimal State Value Function and Optimal Policy 91
4.10 State Value Error 92
4.11 State Value Update 92
4.12 State Value Error for $TD(\lambda)$ with Neural Networks 93
4.13 Eligibility Trace Update for $TD(\lambda)$ with Neural Networks 93
4.14 State Value Function for $TD(\lambda)$ with Neural Networks 93
4.15 Error Term Computation for SARSA and QL 93
4.16 Action Value Update for SARSA and QL 93
4.17 Error Term Computation for SARSA and QL with NNs 94
4.18 Action Value Update for SARSA and QL with NNs 94
4.19 Policy Gradient Objective Function Initialization 94
4.21 Policy Gradient Objective Function Gradient 94
4.22 Policy Gradient Objective Function Update 94
4.23 Monte Carlo Objective Function Update (REINFORCE) 95
4.24 Q-Value Actor-Critic Objective Function Gradient 95
4.25 Q-Value Actor-Critic Objective Function Update 95
4.26 DQN Action Selection 96
4.27 DQN Target Network Soft Update 97
4.28 AZ MCTS Node Selection Puct Heuristic 101
4.29 AZ MCTS Edge Value Update 101
4.30 AZ Loss Computation 102
5.1 Least Utilized Downstream Machine Heuristic 109
5.2 Jm Precedence Constraints 110
5.3 Jm No Overlap Constraints 110
5.4 Jm No Preemption Constraints 110
5.5 FJc No Overlapping Work Centers Constraints 110
5.6 FJc Routing Alternatives 110
5.7 $FJc|M_i^o$ Routing Alternatives 110

5.8 Minimum Makespan Objective . . . 111
5.9 Time Dependent Operation Set . . . 111
5.10 Time Dependent Job Set . . . 111
5.11 CP3 Clipped Operation Set . . . 112
5.12 SimSearch Worse-Case Runtime . . . 115
5.13 MCTS Node Selection Criteria . . . 118
5.14 MCTS Backtracking Value Update . . . 118
6.1 WIP Slot Release Time Distribution . . . 128
6.2 Interarrival Time Distribution . . . 128
6.3 Release . . . 128
6.4 Utilization Difference Reward . . . 133
6.5 Quantized Utilization Difference Reward . . . 133
6.6 Utilization Sigmoid Reward . . . 133
6.7 Time-Scaled Utilization Difference Reward . . . 133
6.8 Quantized Utilization Deviation . . . 133
6.9 Negative Absolute Makespan . . . 133
6.10 Duration Relative Makespan . . . 133
6.11 Negative Cumulative Buffer Lengths . . . 133
6.12 Illegal Action Punishment Extension for Negative Absolute Makespan . . . 147
6.13 Virtual Best Selector Makespan . . . 152
6.14 Virtual Best Selector Relative Makespan . . . 152

Chapter 1

Introduction

"Begin at the beginning", the King said gravely, "and go on till you come to the end: then stop."

— Lewis Carroll, Alice in Wonderland

The importance of scheduling cannot be overstated. It is ubiquitous across human activity in general, and across scientific fields and industrial applications in particular. Technology-enabled solutions for complex scheduling problems are all around us, from planning hospital shifts, through deciding upon the next process being allotted the CPU, to deciding which machine is to process the next operation required for the construction of a product.

Since the advent of industrialization, production has taken up a central role in our societal environment. As such, production scheduling stands out among other fields. Advances herein have a tremendous, albeit not always obvious, impact on human life. In an industrial production setting, planning the assignment of valuable resources, such as human labor or machine processing time, is crucial. A good production scheduling solution can lead to higher monetary gains and an increased customer satisfaction through a more efficient resource use. Given the elevated profit margins that a good production scheduling scheme promises, the amount of attention the problem garners is understandable.

The field of production is characterized by interdisciplinarity and a high degree of dynamism, given that production reality changes constantly, pushed by digitalization trends. As production data and computing resources become more readily available and production itself becomes more versatile, the incorporation of Machine Learning (ML) techniques, particularly Reinforcement Learning (RL), for production scheduling becomes more attractive.

Broadly speaking, production scheduling is the problem of sequencing a number of operations that are associated with different jobs onto production resources, such that an objective function is optimized. Different constraints on the set of jobs and production resources along with different relevant goals define individual scheduling problems. Production scheduling is NP-complete for most real-world cases, and, as such, difficult

to solve optimally for large production instances. Furthermore, unexpected events such as new job arrivals, operation duration deviations, or resource availability issues, may invalidate schedules, leading to a need for their re-computation.

Solutions to production scheduling problems mainly fall into one of three categories, namely exact approaches, priority rules, and meta-heuristics. Exact approaches, e.g. Mixed Integer Linear Programming (MILP), Constraint Programming (CP), seek to find the optimal solution with respect to the objective function. Priority rules define preferences for operations that are to be assigned to resources, e.g. prefer operations with the Longest Processing Time (LPT), and, by design, do not yield optimal solutions. Meta-heuristics, e.g. Evolutionary Algorithm (EA), RL, also defer optimality in favor of good enough solutions that are found through different flavors of non-exhaustive solution space search. Note that using the term "exact approches" in stochastic environments is a misnomer, since optimality can only be defined ex-post, and not ex-ante. However, assuming the absence of stochasticity and employing a re-planning strategy, exact approaches are very much still applicable, often with *good*, albeit *not (necessarily) optimal* results. For lack of a better term, we continue using "exact approaches" to refer to the category of solution approaches it encompasses.

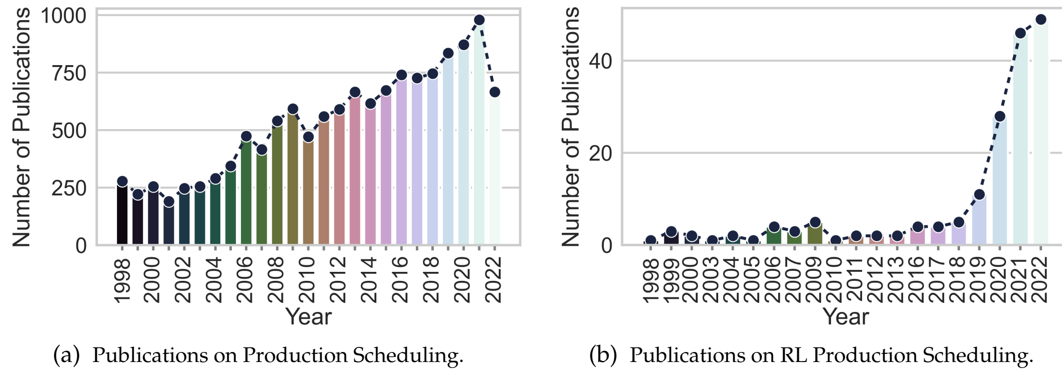
RL is broadly defined through the interaction between an agent, and its environment. The agent takes actions within the environment based on the perceived state and receives feedback by means of a reward signal. Its task is to take actions in such a way that his cumulative reward is maximized.

RL schedulers are fast, require no mathematical modeling (as opposed to exact approaches), promise a high degree of adaptivity and could transfer learned patterns between different production setups. Adaptivity, which we define as the capability of maintaining solution quality under information uncertainty (see Chapter 4 for a more detailed discussion of the term), is especially important since the production environment is highly complex with the planning input being characterized by uncertainty. These advantages make the undeniable interest in RL solutions (see Figure 1.1) understandable. While the interest in scheduling topics seems to grow linearly, the interest in RL scheduling solutions displays an expponential like behavior with the number of publications on the topic rising from less than five between 1998 and 2018 to more than 40 in 2022.

1.1 Problems in the RL Production Scheduling World

While growing in volume, the RL production scheduling work, is still young and suffers from several problems stemming from a lack of standardization. Aside from the missing standardization itself, these problems are the absence of a baselining production simulation framework, the lack of RL-competitive baseline scheduling algorithms and the absence of focus with respect to scheduling setups.

There are three dimensions to the **standardization problem**. First, no unified way of discussing production scheduling setups exists. This gives way to ambiguity in the problem descriptions impeding clear conclusions in experimental work. Production scheduling



(a) Publications on Production Scheduling. (b) Publications on RL Production Scheduling.

Figure 1.1: Distribution of publications on the topic of production scheduling problems between 1995 and 2020 (left). Distributions pertaining to reinforcement learning solutions for production scheduling problems between 1995 and 2022 (right). Numbers were compiled using the Web of Science Portal using the topic search with the keywords "shop scheduling" (a) and "shop scheduling *and* reinforcement learning (b). Date: August 2022"

is not a single problem, rather it is a family of related problems which grows to keep up with the shifting production reality. Secondly, there are many flavors of RL, and, more importantly, many ways in which RL can be modeled to solve production scheduling problems. While for problems like chess the definition of RL elements (action, states, etc) is fairly intuitive, the same cannot be said about production environments. Finally, and most importantly, there is no standard validation practice in RL scheduling literature. Experiments are often not reproducible and insufficiently baselined.

The growing body of RL scheduling literature displays an increasing sophistication of RL solutions to increasingly complex production scheduling problems but the RL potential is still hard to assess within the field. While in the past relatively simple Q-Learning (QL) approaches were the norm, nowadays varied novel RL algorithms such as AlphaGo Zero (AZ) and Deep-Q Networks (DQN) built on top of increasingly complex network structures ranging from Fully Connected Neural Network (FCNN), to Graph Convolutional Network (GCN), and Self Organizing Maps (SOM). Despite the hundreds of papers on the subject, the lack of standardization makes a conclusive statement on the position of RL within the scheduling context difficult.

Our target field could benefit from the established evaluation approaches from more traditional ML fields such as computer vision. Here, the state of the art is more easily established, because of the standard benchmarking datasets available, such as ImageNet, Cifar, etc. Authors run their experiments on these common, publicly available datasets, and report their results using standardized evaluation techniques. As such, improvements on the state of the art are mostly transparent.

To validate RL approaches a currently unavailable general and RL compatible **benchmarking simulation framework is required**. RL is intrinsically linked with simulation. Since it would be too costly and time-consuming to train agents in a live environment, a simulative environment becomes the only viable alternative. Additionally, simulations are required to test the influence of stochastic factors on different solution approaches. Broadly, an RL

compatible simulation is one that allows RL agents to guide its execution by means of actions (agent outputs) while providing them with a state (agent inputs) representation and a reward signal reflecting the quality of their decision making. This is most easily achieved by implementing the OpenAi Gym Application Programming Interface (API) (Brockman et al., 2016).

A benchmarking simulation framework would serve three purposes. First, such a framework would guarantee experiment reproducibility. Experiments currently ran on stochastic environments are impossible to reproduce exactly, which requires researchers to additionally implement the models described in the original papers. By the inherent nature of stochasticity, the results reported in literature might not match the own results. Voluntary or involuntary cherry picking (Morse, 2010) may invalidate established results, when not enough experiments were ran. Secondly, the framework would shift focus away from software engineering and towards algorithmic solution engineering. Since the code associated with a simulation experiment is (almost) never available, in order to validate a new scheduling approach for a particular setup against a published RL solution, researchers have to re-implement the simulation. This is indeed a daunting, not to mention redundant, task given all the complexity of production setups simulated.

The frequently employed **RL baseline algorithms are not competitive**, meaning that they mostly do not share the properties that make RL attractive. This yields a baselining research gap. However, meta-heuristics sharing at least the adaptivity, and fast runtime qualities both do exist and can be constructed. RL approaches are generally evaluated by means of comparison with simple priority rules, and, in deterministc scheduling setups, with the known optimum. While adaptive, the priority rules solution quality can be found to be lacking. Conversely, exact approaches that can be used to find optimal solutions in deterministic cases are slow and considered not to be adaptive. Looking towards RL alternatives is important because RL systems incur a high degree of "technical debt" (Sculley et al., 2015). This means that post-deployment, the maintenance of an RL scheduling system could become difficult and costly. Problems can arise from data dependencies, configuration issues, and changes in the real world. Moreover, the authors identify reproducibility among other areas of ML-related debt.

From an application point of view, RL scheduling literature suffers from a **lack of focus on the production scheduling setup**. There is no sine qua non solution approach for production scheduling. The solution requirements depend highly on the characteristics of the production setup considered. In cases where stochastic influences are expected to be only slight and there is sufficient time available for planning and the scheduling problem instance size is manageable, re-planning using exact approaches is preferable to any other approach. Conversely, in highly stochastic cases, using priority rules or even random strategies may yield as good a result as any other approach. At this point, the burden of proof still lies with the research community to identify the exact situation where RL solutions are preferable to others.

1.2 Demystifying RL Production Scheduling Approaches

Because of the problems discussed in the previous section, RL production scheduling approaches are currently shrouded in a certain mysticism. This work intends to render RL scheduling approaches less mystical. Our work serves a dual purpose of creating an experimentation standard for RL approaches to production scheduling and drawing attention to the disadvantages of RL while providing competitive alternative solution approaches.

Our demistification path provides four large (bold numbers) and one marginal contribution to the research community:

1. First, we create an **experimentation standard based on a systematic literature review** of unprecedented breadth and depth in the field in Chapter 2. We isolate and categorize experimental RL production scheduling literature focusing on three aspects, namely scheduling setups, RL modeling, and experiment evaluation. This contribution simultaneously documents the current research gaps.

2. Secondly, in Chapter 3 we **derive requirements for an RL benchmarking simulation framework and provide an open-source implementation** that satisfies them. Our framework, which we named FabricatioRL, is general, configurable, runtime efficient, extensible and guarantees experiment reproducibility.

3. Thirdly, we concisely present the RL theory necessary for the in depth understanding two popular RL algorithms, namely Dual Deep-Q Networks (DDQN) and AZ in Chapter 4. The chosen RL algorithms are elaborated in detail, such that their implementation becomes transparent. The **complete introduction of the DDQN and AZ from theory to implementation** provides a relevant, albeit more marginal contribution to the field.

4. Fourthly, we **identify the relevant criteria for RL-competitive baselines and select or construct the corresponding algorithms** in Chapter 5. We provide a total of three novel baselines two of which were created by us. The first baseline algorithm — Simulation Search (SimSearch) — emulates RL but relies on simulation instead of prediction to make informed decisions. The second baseline — CP3 — is a adaptation of CP which defers optimality in favor of faster runtimes. The third baseline — Monte Carlo Tree Search (MCTS) — was selected from the broader scheduling literature based on its characteristics.

5. Finally, we put everything together, thus providing an exemplary set of **fully reproducible experiments with a distinct focus on both production setup and RL design** in Chapter 6. The experiments compare the performance of DDQN and AZ against our baselines on two widely encountered stochastic setups, namely a dynamic job-shop scheduling problem ($Jm|r_j^s|C_{\max}$) and a dynamic flexible job-shop scheduling problem with machine capabilities ($FJc|r_j^s, M_i^o|C\max$). Before the final evaluation, a model selection phase is used to assess the quality of several RL models varying both design and model parameters. When constructing the scheduling

setups, we use FabricatioRL to determine key environment parameters. We limit the extensive design choices by means of a novel ex-ante evaluation of (feature-)state representations and reward signals.

Our experiments confirm some of the results in literature while simultaneously relativizing the RL performance with respect to both solution quality and transferability, and highlighting the importance of scheduling setup analysis. In the FJc case AZ marginally outperforms the priority rules baselines often used in literature. However, the baselines we propose significantly outperform AZ. Additionally, our DDQN approach does not outperform all priority rules. Thus the high RL performance reported in literature is relativized. The transfer learning capability often used as an argument for RL is also relativized given that when deployed to the Jm setup, our RL approaches, which were trained solely on the FJc setup, perform poorly. By carefully assessing the job-arrival behaviour and relating it to the scheduling results, we demonstrate that the scheduling algorithm choice should be based on the setup at hand. Within the FJc setup, which offers more optimization potential, the performance difference between different scheduling approaches is much higher than within the Jm setup.

While this work is quite extensive, there are some aspects which we do not consider so as to keep within its frames. Firstly, we do not review the entirety of the RL literature available. Rather we target the most influential papers currently known to us. With respect to the categories defining the elements of our standard, we focus on high level aspects rather than exact details. For instance we note down whether actions are direct or indirect rather than specifying the exact variable significance, e.g. job indices or particular priority rules. Similarly we discuss whether approaches were sufficiently baselined rather than listing the algorithms employed. Secondly, we do not provide all implementation details of neither our simulation framework nor our algorithms. Rather we describe everything in such a fashion that the reader can more easily dive into our code, which we open sourced. Thirdly, we cannot investigate all the setup-RL design-baseline combinations encountered in literature. Instead we focus on popular setups and designs coupled with baselines that are RL-inspired and/or RL-competitive.

Please note that, given the large number of variables required to explain the different elements of our work, the symbols we use are not always unique throughout this elaboration. We do our best to respect scheduling literature naming conventions and ensure uniqueness of variables used between sections. However, the reader is still advised to contextualize variables and mathematical symbols locally rather than globally.

The present elaboration both bundles and extends several of our past papers and serves as a root for future publication effort. Chapter 2 builds upon the conference paper titled "Towards Standardizing Reinforcement Learning Approaches for Production Scheduling Problems" (Rinciog et al., 2022), which was initially published as a white paper (Rinciog et al., 2021c). Chapter 3 is based on paper we called "Fabricatio-RL: A Reinforcement Learning Simulation Framework for Production Scheduling" published in the Winter Simulation Conference (Rinciog et al., 2021a), that details the first version of our software (Rinciog et al., 2021b). The description of AZ in Chapter 4 closely follows the one put

forward in "Sheet-Metal Production Scheduling Using AlphaGo Zero" (Rinciog et al., 2020). The double blind peer-review process associated with three of the five listed publications (Rinciog et al., 2020; Rinciog et al., 2021a; Rinciog et al., 2022) served confirm the need this thesis. Future experiment papers using the original work in Chapters 5 and 6 are currently being developed alongside a journal paper detailing the state of the art in our field based on Chapter 2.

Chapter 2

Ordering Disorder: A Standardization Framework for RL Production Scheduling

If you think of standardization as the best that you know today, but which is to be improved tomorrow, you get somewhere

— Henry Ford

This section contains a meta-analysis of published RL production scheduling approaches aimed at establishing an experiment design standard that eases reproducibility and boosts the validation strength of future research. To that end, 98 publications from the field are reviewed in terms of their simulated production setup and their RL design. We also assess the reproducibility of the surveyed literature as well as the way in which the results were validated.

The 98 papers to be reviewed were compiled in seven steps. Firstly, we compiled a list of 176 publications from the field by means of a topic search on the Web of Science Portal (WoS) portal using the "shop scheduling" in conjunction with "reinforcement learning" keywords. The choice of our first keyword is motivated by the fact that most canonical production scheduling setups include the term "shop". Out of nine machine setups introduced by Pinedo (2012), five contain the word "shop". Furthermore, the production grounds are widely referred to as "shop-floor". In a second step, we sorted the works descendingly by the number of citations and the publication year. Our third step consisted of eliminating the works with no citations, leading to a new total of 123 works. The fourth step consisted of eliminating 25 publications that did not pertain to production (e.g. Zhang et al., 2017; Yoshida et al., 2009; Park et al., 2020a) eight papers that were conceptual in nature rather than experimental (e.g. Ey et al., 2000; Serrano Ruiz et al., 2021; Serrano-Ruiz et al., 2022) and eight that were supposedly on the topic of RL but did not follow the Markov Decision Process (MDP) formalism (e.g. Kim et al., 1998; Yu et al., 2006). We eliminated five of the remaining papers due to insufficient access rights as a fifth step.

The remaining two steps, while not reproducible, serve to alleviate the potential bias associated with a single search on a single platform. In a sixth step, we expanded our search using google scholar using the following keywords prepended by "reinforcement learning": "shop scheduling", "sheet metal", "semiconductor", "production", "scheduling". For each of the five searches, we selected the first 20 results and pooled them together to form a population of 60 additional papers. Finally, in our seventh step, we sampled papers one by one without replacement, adding them to the papers to be reviewed or discarding them either if they were already contained in the WoS group.

Our contribution of bringing order to the 30 years of work done in the field of RL production scheduling, can be broken down as follows:

1. Firstly, we index the setups used in RL experiments using the Graham notation extending the standard whenever necessary (Section 2.2). The setups are defined as the combination of base scheduling problems, α, additional constraints β and optimization goals γ.

2. Secondly, we index the RL design choices taken by different publications (Section 2.3). The index dimensions are given by the MDP Breakdown, a concept we introduce to group similar modeling approaches on a high level, several fields dedicated to the chosen agents, and information on the type of states and actions.

3. Finally, we assess the validation approaches (Section 2.4) taken by authors with respect to reproducibility and evaluation underlining the respective research gaps.

Our literature based standardization work is framed by the related efforts and research gaps sections (2.1 and 2.5 respectively). We begin by noting down the related effort with respect to the RL scheduling literature investigation and standardization thereby demonstrating the meta-analysis gap. The concluding section lists the gaps that our investigation uncovered, which we strive to address in this thesis.

With respect to standardization section structure, our work follows the same pattern for each of our contributions. First, we briefly introduce structure we used to tabulate the 98 publications with respect to the respective section criteria (setup, design, validation). Then we discuss the different table fields one by one, using examples from the literature. Each field discussion includes a frequency analysis of the different values in that particular column. This provides a useful tool for researchers in the field for isolating the most closely related work from the hundreds of available publications. The full tables, which act as the source for the frequency analysis, are attached in Appendix A.

Note that it would be impossible to losslessly condense decades of research in the current work. It follows that not all aspects of our subject papers can be scrutinized. We do not, for example, look in detail at every piece of information used in the state, action, and reward formulation. We do however give examples of such. Similarly, we do not consider every RL algorithm in the same level of detail. Instead, we focus on the most pervasive agents.

2.1 Related Efforts

Standardization Work: Bartz-Beielstein et al. (2020), give eight criteria for benchmarking within the Operations Research (OR) field, namely clearly stated goals, well-specified problems, suitable algorithms, adequate performance measures, thoughtful analysis, effective and efficient (experiment) designs, comprehensible presentations, and guaranteed reproducibility. Moving past the reproducibility that is in no way guaranteed for production scheduling RL experiments, the absence of standardization affects the rest of the criteria, particularly well-specified problems and suitable algorithms.

In terms of describing production scheduling problems, a notation system respected as canon in much of the scheduling community exists. Being introduced by Graham et al. (1979), the notation system is often called "Graham" or "$(\alpha|\beta|\gamma)$" notation. The first position of the triple defines the base scheduling setup and encodes information about operation precedence constraints as well as the number and type of resources. There can be exactly one α parameter per setup. The β parameters define additional setup constraints, such as transport or setup times. Setup definitions can contain any number of β-parameters, including none. The gamma parameter defines the optimization goal. This Graham notation was extended by Lawler et al. (1993), Knust (2000) and Pinedo (2012). Using $(\alpha|\beta|\gamma)$ one can not only give precise information about the setup at hand but also relate problems to one another, since a subscription relationship is defined over them, with more specific setups subsuming more general ones. However, the notation system needs to be extended to cover the increasingly complex problems tackled by the RL scheduling literature.

First steps in the direction of RL project standardization were taken by OpenAI Gym[1], whereby a general RL API is defined. The Gym API has become the de-facto standard for RL environments with libraries such as keras-rl adopting it for the therein implemented agents. This is also reaffirmed by Hubbs et al. (2020). The authors created OpenAI Gym simulations for varied combinatorial optimization problems from the field of OR, e.g. the Bin Packing or Traveling Salesman.[2]. For the production scheduling problem of the semiconductor industry an environment built on top of tensorforce (Kuhnle et al., 2017), which respects the OpenAI Gym API[3], is provided by Kuhnle et al. (2019). While this is a decisive step forward for RL production scheduling experiments, the environment is fixed to the semiconductor industry material flow. A more flexible setup is required. Another caveat of this implementation is the absence of guaranteed reproducibility.

Past the standardization of agent-environment communication, RL production scheduling literature often suffers from the absence of a clear presentation of the RL design choices. A good standard of this is put forward by Kuhnle et al. (2019). Aside from the RL algorithm itself and its hyper-parameters, a clear definition of the RL-environment interaction loop,

[1]https://gym.openai.com/

[2]The code is available on https://github.com/hubbs5/or-gym

[3]Tensorforce provides tensorflow implementations of several RL agents as well as adapters for several RL environments such as OpenAI Gym, OpenAI Retro, VizDoom etc. and is available at https://github.com/tensorforce/tensorforce

state-space, action-space as well as the agent reward is needed.

Apart from an available simulation code and a well-described RL design scheme, the production setting needs to be clearly defined and the associated experiment inputs need to be made available. There are four families of production scheduling problems traditionally tackled by the OR community, namely the Flow Shop Scheduling Problem (Fm), Job Shop Scheduling Problem (Jm), Flexible Jm (FJc), and Open Shop Scheduling Problem (Om). The distinction criteria for these are given by the technological constraints of operations in a job. Fms contain identical jobs with their operations being totally ordered. At the other end of the spectrum, Oms contain different jobs with no ordering constraints. Jms and FJcs lie between the two. While in their standard formulations these problems are deterministic, stochastic influences, such as job release times, processing time noise or machine breakdowns, can be added to them. For standard deterministic problems, benchmarking inputs are available from previous publications. Jm benchmarking instances were provided by Yamada et al. (1992), Storer et al. (1992), Applegate et al. (1991), Lawrence (1984), Fisher (1963), Demirkol et al. (1998), Adams et al. (1988), and Taillard (1993).[4] Some Fm, Jm and Om instances can be found in Beasley's OR library (Beasley, 1990).[5] FJC instances were provided by Barnes et al. (1996), Brandimarte (1993), Dauzère-Pérès et al. (1998), and Hurink et al. (1994). All of these instances are available through Mastrolli (1998).

These standardization efforts fail to address the particularities of the RL production scheduling field. Firstly, the RL literature defines many scheduling setups using inconsistent nomenclature. Secondly, RL offers many design choices for production scheduling problems. Finally, in stochastic settings, validation is difficult because of the high instance variance. Our work addresses these issues by providing a unified way of discussing scheduling setups, detailing RL design and establishing an RL-specific validation scheme.

Literature Reviews: The works by Jones et al. (1998), Slotnick (2011), Cunha et al. (2018), Mohan et al. (2019), and Kayhan et al. (2021) contain related investigations of the production scheduling literature. Save for Slotnick (2011) and Kayhan et al. (2021), these elaborations are somewhat superficial and serve more as an orientation for researchers seeking to broadly familiarize themselves with the field. Jones et al. (1998) focus on the different method categories, such as the aforementioned mathematical programming or heuristics, for job shop scheduling. Cunha et al. (2018) compare deep reinforcement learning techniques with genetic algorithm approaches for Job Shop Scheduling. Mohan et al. (2019) categorize the types of algorithms employed as solvers for dynamic versions of job shop scheduling problems.

Slotnick (2011) investigates the different scheduling problems solved in literature and creates a taxonomy thereof. The authors split the 70 production scheduling papers with a distinct focus on order release into a conceptual category (A) consisting of four papers, and four experimental categories namely "deterministic single-machine problems" (B),

[4]The shop instances extracted from all of these publications as well as known upper and lower bounds for them in terms of makespan can be found here: http://jobshop.jjvh.nl/.

[5]http://people.brunel.ac.uk/~mastjjb/jeb/orlib/flowshopinfo.html

"stochastic single-machine problems" (C), "multiple-machine problems" (D), and "order rejection problems" (E) consisting of 22, eight, 21, and 15 papers respectively. Within each experimental category, the authors investigate the different setup characteristics present in the respective papers as well as the optimization objective and the solutions deployed, e.g. heuristics, MILP, and EA. The setup characteristics taken into account by the authors reveal themselves to be the type of arrival times (stochastic, deterministic, or, in the case of E, a variable to be optimized), alongside 8 additional flags indicating the presence of resource setup times, preemption, a pricing model, release dates, deadlines, adjusted processing times, precedence constraints and resource constraints. Note that not all flags are present in each experimental category.

While the authors provide a good overview of problems considering order release, the present setup flags do not clearly relate to the canonical Graham notation. As such the present categories and the relationship between them become somewhat murky. Furthermore, the solution categories listed differ in level of granularity, with broader categories such as "heuristics" or "neural" being sometimes used, while at other times a more precise approach designation such as MILP or EA is given. Finally, given the enormous volume of work in the field (see 1.1) the elaboration is, by now, somewhat dated. Different from the work of Slotnick, 2011, we focus on RL scheduling, describe the scheduling setups using Graham notation, investigate validation on an high level consistently and consider recent papers in our analysis.

The work most closely related to ours was done by Kayhan et al. (2021). Herein the authors investigate 80 scheduling papers containing RL scheduling approaches with respect to eight aspects. The "problem type" (1) describes the scheduling setup using the Graham notation, with extensions thereof whenever necessary, e.g. HF for the hybrid flow shop, and includes the optimization criteria. The "objectives" (2) field details whether the indexed works employed single- or multi-criteria optimization schemes. Using "system type" (3), the authors distinguish between deterministic and stochastic problems. The authors' "learning algorithm" (4), "action selection" (5), and "state definition" (6) criteria all pertain to components of the RL algorithm albeit in a somewhat unusual way: (5) tabulates the exploration strategy rather than the action-spaces and (6) indexes the agent-internal state representation (e.g. neural network) rather than the expected state-space. The "learning algorithm" column indicates the RL algorithm itself, e.g. Q-learning, temporal difference learning. "Action selection" refers to the exploration strategy employed, e.g. ϵ-greedy, Boltzmann probability. "State definition" refers to the type of RL function, e.g. neural network, aggregation (i.e. table). The "agent" category (7) signals whether the investigated works deploy single or multiple RL agents. Finally, using the "benchmark method" category (8), the authors note down the algorithms used to compare the RL approaches' performance.

Upon closer inspection, the fields used to structure the authors' comprehensive review reveal some caveats with respect to the scheduling problem (1, 2, 3), RL (4, 5, 6, 7), and validation approach (8). Firstly, the extensions introduced for the "problem type" field fail to take the work of Pinedo (2012) into account leading to duplication at times, e.g.

in the case of the hybrid flow-shop (Ruiz et al., 2010; Armstrong et al., 2021), which is well defined as a flexible flow shop in Pinedo's work. Furthermore, the subsumption relationship between different constraints is not clearly delineated. In terms of the "system type" category, the authors are at times too trusting of the investigated works. Zhao et al. (2021c), for instance, motivate their use of RL using the stochastic nature of production as a pro-RL argument, however, they run their experiments on a deterministic setup. Yet Kayhan et al. (2021) index the setup employed by Zhao et al. (2021c) as stochastic. Secondly, the fields used to detail the RL algorithms fail to capture the RL design, i.e. the state, action, and reward-spaces. Thirdly, the validation approach used in the investigated works should be extended to account for reproducibility and a sufficient number of experiments having been run. Our work distinguishes itself from the one lain down by firstly matching the scheduling setup with Graham categories by closely inspecting the constraints detailed by the different authors rather than their own naming of the setup. Secondly, we clearly report the RL designs, extending the recipe by Kuhnle et al. (2019). Finally, we take reproducibility into account as well as the baseline algorithms.

It is also worth mentioning, that the applied methodology for the full paper read selection may contain a bias towards Q-Learning and tabular RL approaches to scheduling which may not reflect current trends in the field. The authors used "Reinforcement Learning", "Scheduling", "Q-Learning" and "Neuro-dynamic Programming" as keywords for an article body search on the WoS portal. Due to the use of the overly specific keyword "Q-Learning" during the literature search, policy-based RL approaches such as Trust Region Policy Optimization (TRPO) or actor-critic methods such as Proximal Policy Optimization (PPO) or AZ may not be captured. We avoid this bias by only using "Reinforcement Learning" in our search string to capture the ML component of the target papers.

2.2 Job Shop Scheduling Variants

In what follows we use an extension of the Graham notation to describe the different production scheduling setups studied in RL literature. We detail the different values of α, β and γ defined by Pinedo (2012) together with the required extensions in subsections 2.2.1, 2.2.2 and 2.2.3 respectively

Table A.1 offers an overview of the deterministic and stochastic production setups studied in the RL scheduling literature in terms of α, β, γ, and the particular maximum problem size. The question mark signals ambiguity pertaining to the respective detail. Several publications (e.g. Jiménez, 2012; Reyna et al., 2015) run experiments on more than one setup. In such cases, we consider the most general setup only. Note that the problem size simply serves as an orientation for the combinatorial complexity tackled by the different publications and is not explicitly discussed within this thesis.

2.2.1 Production Scheduling Problem Families

Formally, production scheduling is the problem of assigning start times $s_{ji} > 0$ to a number of operations o_{ji} with processing times p_{ji} grouped into n jobs ($J \in \mathcal{J}$), onto one of m production resources $M \in \mathcal{M}$ to optimize some target value, for example makespan which is defined as $C_{\mathbf{max}} := \max\{C_j : j \in \{1 \dots \mid J \mid\}\}$, where $C_j := \max\{s_{ji} + p_{ji}\}$ is the completion time of job with the index j. There are two more variables associated with jobs, namely, release dates r_j and due dates d_j. The release date specifies the point in time from which the job can be processed ($\forall j \forall i : r_j \leq s_{ji}$), while the due date is the point in time before which the job needs to be finished. Jobs finished past the due date generally lead to some form of quantitative or qualitative penalization for the production site. In what follows, if not otherwise specified, we use j to denote a job index and i for machine indices. In most cases, resources are limited to processing one job at a time (no-overlap) and cannot be interrupted (no-preemption).

The main types of production resources are processing and transport resources, which we henceforth refer to as machines and vehicles respectively. Note that the machines and vehicles terms are, strictly speaking, misnomers. Processing resources execute job operations and can be either human laborers, machines or a combination of both. Transport resources transfer jobs from one processing resource to another. Again, transport resources can be either humans, human operated vehicles, automated vehicles or any other transport technology (coveyors, cranes etc.). However, the chosen terminology both better aligns with the bulk of the scheduling literature and is more concise.

Figure 2.1 shows the hierarchy of scheduling problems aggregating the values of the α-column from Table A.1. The green fill color indicates that the problems were defined by Pinedo (2012), while the red fill color indicates that they were defined by us to accommodate setups found in RL scheduling literature. Problems on gray-filled rectangles are logical extensions of present problems that have yet to be studied. Numbers in parentheses pertain to the number of publications that used the respective setup. The generalization relationship between problem classes is indicated with arrows.

The α parameter takes exactly one of the following categories as a value. Depending on the number and order of operations within jobs and the number and speed of resources available for different operation types, production problems can be categorized into one of 17 problem classes. Figure 2.1 displays two generalization dimensions starting with single machine problems, 1, in the lower left corner, and ending with the unrelated flexible partially open shops in the top right corner, $Rm + FPOc$, which is the super-class of all setups at hand. The first generalization dimension (left to right) differentiates classes based on number of machines and their speed, while the second generalization dimension (bottom to top) differentiates setups based the job operation types and the operation order within the jobs.

Single machine problems are defined in terms of n jobs, each containing exactly one operation with duration d_{j1}. The task is to schedule all operations onto a single resource. If we extend this setup by replicating the machine we obtain the Pm setting. By endowing

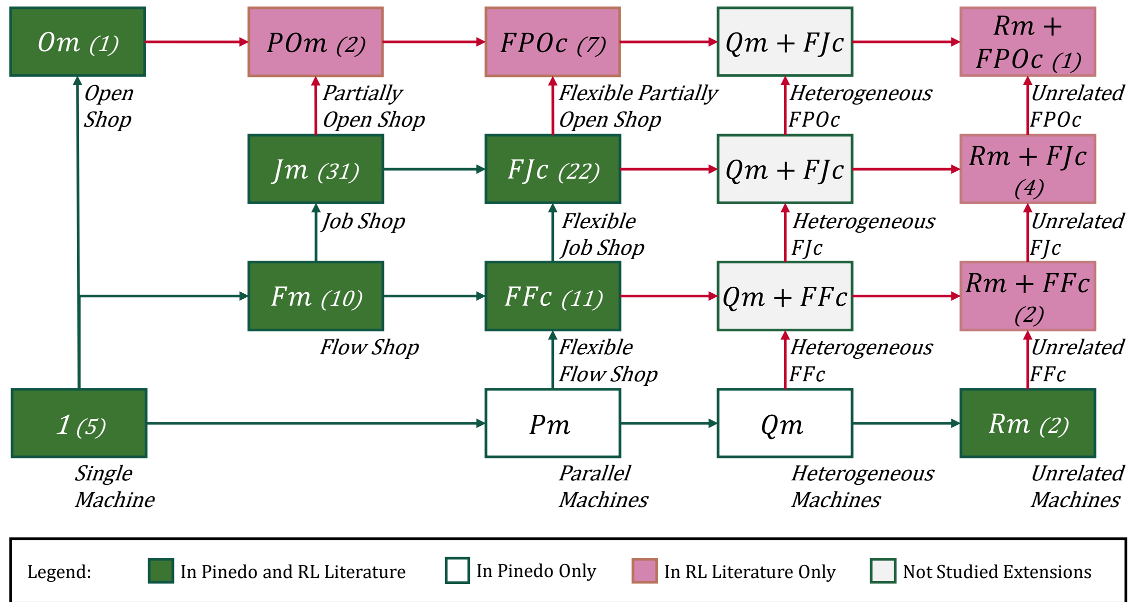


Figure 2.1: Hierarchy of machine setups (α) for scheduling problems. The number of publications containing the particular setups are noted in parentheses. The entries in green were selected from the work of Pinedo (2012). The hierarchy needed to be extended to cover the scheduling problems reviewed (red boxes). The gray filled boxes represent as of yet unstudied logical extensions noted down for completion. The Problems on a white background have yet to be studied in relationship with RL. The arrows represent the generalization relationship.

resources with different speeds v_i we reach the Qm setup. The processing time of a particular operation o_{ji} is, in this case, dependent on both its base duration d_{ji} and machine i's speed v_i, i.e. the total duration will be $d_{ji} \cdot v_i$. If the machine speed depends not only on the machine index i but also on the job index j, the production has an Rm setup.

By extending the 1-setup to contain m resources with jobs consisting of the same number of operations, we obtain "shop-scheduling" problems. Each operation is bound to a particular resource, and each resource is visited exactly once to complete a job. We use the mapping $type : \mathcal{O} \rightarrow \mathcal{M}$ to denote the machine binding of the particular production scheduling problem, where $\mathcal{O}$ is the set of all operations over all jobs and $\mathcal{M}$ is the set of all machines. If operations are totally ordered, i.e. the constraint $\forall j \in \{1, \ldots, n\} : \forall i_1, i_2 \in \{1 \ldots m\} : i_1 < i_2 \Rightarrow s_{ji_1} < s_{ji_2}$ is in effect, and all jobs need to be processed on machines in the same order, then the production setup is that of Fm. If job operations are totally ordered, but the machine sequence is different for different jobs, then a Jm is in place. If job operations have no precedence constraints, then an Om is in place.

Any shop setup can be extended by replicating one or more resource types and grouping them into so-called "work-centers". The $type$ mapping now gives the index i of the work-center c_i where the operation can be executed. Note that an operation can be executed on machines of exactly one work-center, i.e. $\forall j : \forall i_1, i_2 : o_{ji_1} \neq o_{ji_2} \Rightarrow type^{-1}(o_{ji_1}) \bigcap type^{-1}(o_{ji_2}) = \emptyset$. If the setup prior replication was Fm, the setup now becomes FFc. Analogously Jm becomes FJm. While a flexible Om is certainly plausible, we

have not found evidence of such setup neither in the RL literature nor in the systematization put forward by Pinedo (2012).

The setup used by Zhang et al. (1995) and Zhang et al. (1996) does not fit with any of the α values described to this point. The two works describe a planning problem where jobs contain only *partially* ordered operations. This means that the precedence constraints for operations in a job can be described by a Directed Acyclic Graph (DAG) (Thulasiraman et al., 2011), instead of a chain, as is the case for the $(F)Jm$ and $(F)Fm$ families. This setup is also more recently employed for final assembly use-cases within the automobile industry (Oh et al., 2022). Since job operations in this setup are partially ordered, we dubbed it partially ordered job shop (POm) and flexible partially ordered job shop ($FPOc$) for the variants with and without replicated machines respectively.

Having briefly discussed the constraints that apply to job operations, we turn our attention to the machines themselves. These are limited to processing one operation at a time, in the absence of preemption.

In Figure 2.1 the rightmost two columns contain setups which combine the different machine-speed setups (Qm and Rm) with the different flexible shop setups (FFc, FJc and $FPOc$) using the + sign. This signals situations where the processing speeds of operations from jobs respecting the structure defined by FFc, FJc and $FPOc$ are machine dependent (Qm+) or job and machine-dependent (Rm+). We used the + sign instead of coming up with new names so as to keep in line with Pinedo's α definitions as much as possible. Note that many of the FJc instances available through benchmark sets (e.g. Mastrolli, 1998) are, not always "pure" FJc setups, as per Pinedo, but, in fact $FJc + Rm$ instances.

Note that in the case of the Pm, Qm and Rm setups, their super classes, as well as Om, and POm, two types of *scheduling* decisions need to be taken, namely *job routing* and *sequencing*. Since in these cases some or all job operations can be executed on a set of different machines, rather than a single machine, the machine assignment for operations becomes a choice variable. Assuming the machine assignment was decided, i.e. the job route has been set for all jobs, the order in which operations get executed by a particular machine needs to be chosen. We refer to the second decision type as "sequencing". The complete set of both decision types defines the schedule.

From the number frequency of setup occurrences in literature, we observe a slight tendency towards more flexible scheduling problems, i.e. those allowing for job routing flexibility. Of the 98 papers investigated, 46 built on setups only requiring sequencing decisions (Fm, Jm, 1), while 52 contained setups with routing flexibility (Om, POm, FFc, FJc, $FPOc$, Rm, $Rm + FFc$, $Rm + FJc$, $Rm + FPOc$), thus requiring both job routing and sequencing decisions. Overall the most popular setups are Jm (31) and FJc (22). This ranking is somewhat reflected by the number of available benchmark instances.

2.2.2 Additional Setup Complexity

The β values modify the scheduling problem by adding constraints or changing existing ones. Figure 2.2 aggregates the information in the β-column from Table A.1. The β-

parameters encountered in literature are presented along with the number of times they were used in different publications. Since multiple β values are possible for a single setup, the total number of parameters need not be equal to the number of publications reviewed. The green fill color indicates that the constraints were both defined by Pinedo (2012), and present in the RL literature. If a rectangle is filled red, then it is part of the setup considered in at least one of the publications surveyed, but not defined by Pinedo. White filled rectangles were defined by Pinedo, but were not present in the surveyed papers. Gray filled rectangles mark logical extensions present neither the work of Pinedo, nor the RL scheduling literature. Arrows indicate an extension/generalization relationship.

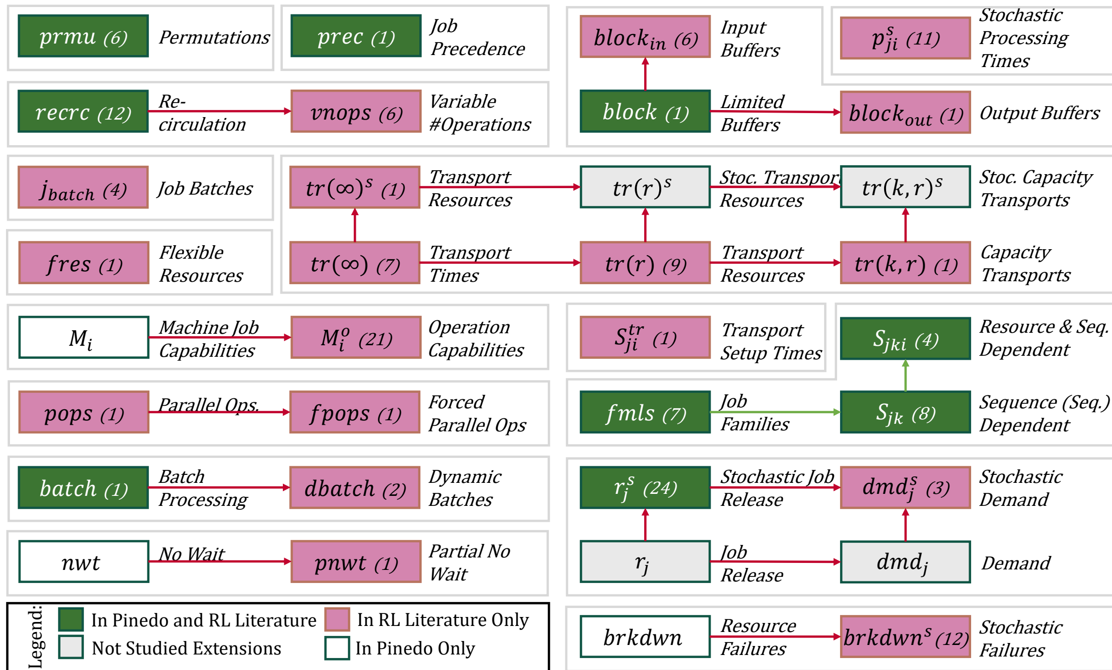


Figure 2.2: Job Properties (β). The numbers in parentheses indicate the number of publications the constraints are present in. The values in the red or green filled blocks can be found in the reviewed literature setups and were defined by us and Pinedo, 2012 respectively. White filled rectangles correspond to parameters defined by Pinedo that are not present in the RL literature. Gray filled blocks are logical extensions we define for completion. Arrows represent a generalization relationship.

The $prmu$ parameter, defined by Pinedo (2012), which is present, for instance, in the works of Wu et al. (2020), Yang et al. (2021b), Yang et al. (2021a), and Lee et al. (2022), stands for "permutations". It can be applied for Fm setups only and indicates that jobs need to be processed on all resources in the exact same sequence. Thus, the sequence in which job operations are processed by the first resource fully determines the schedule, meaning that solutions can be represented as a permutation of the n jobs to be processed. This considerably reduces the solution space.

The "Precedence" parameter $prec$ introduced Pinedo, 2012 defines additional precedence constraints, though this time over the set of *jobs* rather than operations. Hereby, a total order $\prec$ is defined not only over operations from individual jobs but also over the job set itself. Jobs operations from jobs j_1 can only be processed on any resource if all the operations from all the jobs j_2 preceding it ($j_2 \prec j_1$) have been completed. $prec$ also

tightens the solution space and was encountered in a single publication, namely in the work of Moser et al. (2020).

An example of a constraint implied by α being changed by a β-value is "re-circulation" ($recrc$). If this value is present in the β set, then job operations are not constrained to be processed on every machine type exactly once. Instead, some work centers can be visited more than once, whereby cycles ensue in the operation precedence graph, hence the name. From the details offered by Pinedo (2012), it is unclear if $recrc$ allows for a variable number of operations in jobs. To model the situation where different jobs require a different number of processing steps (e.g. Zhang et al., 1996; Zhang et al., 1995; Rinciog et al., 2020) we introduce the more general constraint relaxation of a variable number of operations per job, $vnops$, which subsumes $rcrc$.

All base setups assume the existence of unlimited buffer spaces between resources. The constraint of limited buffer space can be added by adding the $block_{in}$ or $block_{out}$ value to β to limit either resource input and/or output buffers to a capacity b_{in} and/or b_{out}. Note that the related $block$ parameter described by Pinedo (2012), does not distinguish between in- and output buffers. The presence of $block$ is defined for Fm and FFc setups only and indicates the presence of buffers between consecutive processing stages. The implication is that a resource can only process a job if there is enough space in the buffer following it. In reality, buffers can be placed either before or after a machine. This distinction is important when there is job routing flexibility as a result of the machine setup. When the input buffer of a machine is full, no job operation can be routed to it until another position gets freed by mapping a buffered operation to the corresponding machine. Conversely, when an output buffer of a machine is full, job operations can be routed to the machine, but no further processing is possible until a buffer position gets freed by routing a job operation to the next processing stage. Setups where $block$ is present are subsumed by both $block_{in}$ and $block_{out}$ defined by us.

Stochastic operation processing times being present in a particular setup are indicated by the p^s_{ji} parameter defined by us. When p^s_{ji} is not present, the processing times p_{ji} of operations o_{ji} are assumed to be deterministic. In reality, however, these times are just estimates, and as such, subject to noise. Some of the authors consider this explicitly (e.g. Wang et al., 2007; Jiménez, 2012; Stricker et al., 2018; Park et al., 2020b).

Whenever present, our extension parameter "job batches" j_{batch} indicates that due dates d_{ji} are specified for groups of jobs rather than individual jobs. The group specification along with the associated due dates are required so as to compute group-level optimization goals such as the total weighted batch tardiness (e.g. Wu et al., 2020). While no other grouping usage was identified in the literature, in theory, j_{batch} could be used in conjunction with other optimization goals, e.g. batch flow time.

To model environments with multiple machines capable of executing an operation type (e.g. FJc, FFc), where the work-center capacity can vary dynamically, we add $fres$ (flexible resources) to the possible β-values. Such an environment was studied by Thomas et al. (2018), where bottleneck resources could be enhanced during the scheduling process.

Another aspect that is traditionally not considered by the OR community is that of transport times and vehicles. When not considering these, two implicit assumptions are made, namely that (a) distances and hence transport times are either negligible, constant or part of the processing times p_{ji}, and (b) vehicles are always available to transport a job immediately after its latest operation has finished.

As soon as (a) does not hold, which can be the case for setups with job routing flexibility, transport times should be modeled separately. If the production setting to be scheduled contains enough vehicles, (b) can still hold (e.g. Thomas et al., 2018). We mark this setup by means of $tr(\infty)$ (transport times) in β. If on the other hand, vehicles are limited, or even a bottleneck resource, we have to model them explicitly in terms of location, since the latter determines the job transport waiting time and is a supplementary scheduling decision. We call the corresponding β-value "Transport Resources" ($tr(r)$). Examples of $tr(r)$ can be found in the experiments conducted by Arviv et al. (2016) and Kuhnle et al. (2020). Finally, some authors model situations where vehicles can carry more than one operation between machines (e.g. Kim et al., 2022). This is signaled our "Capacity Transports" extension $tr(k, r)$.

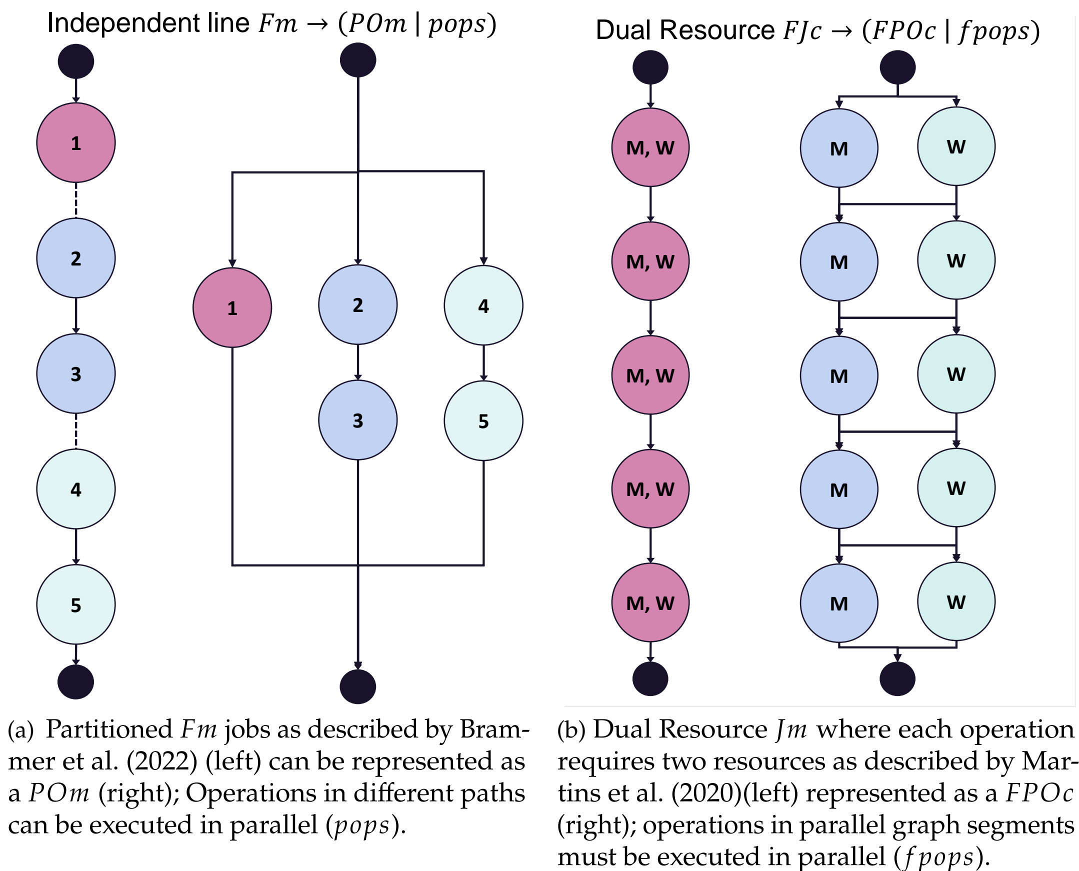
For each of the vehicle parameters, a stochastic version could be defined, if the transport times are assumed to be very noisy. While such an assumption was only encountered once in the surveyed literature (Han et al., 2019) we introduce the extension parameters $tr(\infty)^s$, $tr(r)^s$ and $tr(k, r)^s$ for the sake of completion.

In all the settings considered up to this point, any particular machine could only execute one type of operation. It could be, however, that a machine can execute multiple operation types (e.g. Martínez et al., 2011; Jiménez, 2012; Bouazza et al., 2017; Luo, 2020). Pinedo defines machine eligibility restrictions M_j as a β value for Pm environments only, and uses it to limit the capabilities of machines of executing any and all operations to a particular subset thereof: $M_i := J' \subset \mathcal{J}, \forall i \in \{1, \ldots, m\}$. Note that since in Pm jobs only have one operation, a job index uniquely identifies an operation. This concept needs to be expanded to cover setups past Pm. We use the M_i^o value in β to distinguish between the original machine eligibility constraints for jobs and our more general ones for operation types. In the M_o context, the sets $M_i := O' \subset type(O)$ for $i \in \{1, \ldots, m\}$ describe the machine capability constraints.

Our "parallel operations" $pops$ and "forced parallel operations" $fpops$ extensions relax the constraint on job operations to be executed one at a time. In setups with a partial operation order within jobs, e.g. Om, POm, $FPOc$, operation precedence can be represented as a DAG, with nodes v, w representing operations and directed edges (v, w) marking that operation w can only be processed after v has been completed.

The presence of $pops$ allows for operations in parallel paths to be executed simultaneously, as long as the precedence constraints are respected. This allows for the definition of the setup described by Brammer et al. (2022) (Independent Line Fm) to conform to the formalism used by Pinedo (Figure 2.3a). In the aforementioned work, the authors segment Fm jobs into operation partitions with no order being defined for individual partitions.

This corresponds to a POm setup. Individual job partitions can be executed in parallel, which is described by the just introduced $pops$ extension. Specifying $fpops$ as a beta value forces operations in parallel graph paths to be executed concomitantly. This makes setups such as that detailed by Martins et al. (2020) ($\alpha|\beta|\gamma$) definable (Figure 2.3b). In these two cases, an extension of α, which is implied by the authors, should be avoided, since α parameters are solely concerned with operation precedence (total, partial or none), machine replication (single machine or work-centers) and machine speeds (homogeneous, machine-dependent or machine and job dependent), none of which are affected by the suggested modifications.



(a) Partitioned Fm jobs as described by Brammer et al. (2022) (left) can be represented as a POm (right); Operations in different paths can be executed in parallel ($pops$).

(b) Dual Resource Jm where each operation requires two resources as described by Martins et al. (2020)(left) represented as a $FPOc$ (right); operations in parallel graph segments must be executed in parallel ($fpops$).

Figure 2.3: Examples of using β-parameters instead of defining new machine setups (α). This should be preferable, since α-parameters are distinguished by job operation ordering (total, partial or none), machine replication (single-machine or multiple machines per operation) and machine speeds (homogeneous, machine-dependent or machine and job dependent), and the new setups do not extend any of these three dimensions.

In many cases, the constraint of sequence-dependent setup times s_{jk} is required for a realistic mapping of the production reality. The s_{jk} constraint, describes a setting where a machine incurs a processing time penalty s_{jk} when switching from an operation from job j to an operation from job k. This could represent the time required to change the necessary tools at a processing station or re-configuring a machine. If the setup times are supplementary machine dependent, the β-value s_{jki} reflects this. Pinedo also describes the situation where jobs can be grouped into so-called families. When switching between jobs of the same family, no setup times are incurred. The aforementioned extension of the setup

times value is marked by the presence of the $fmls$ value in the β section of the problem definition. Similarly, vehicles can incur a penalty for switching between transporting different jobs. This is expressed by the extension parameter s_{ji}^{tr}.

A further example of a constraint relaxation is the $batch$ parameter. If this value is included in β, machines are allowed to process multiple operations simultaneously. Batches are fixed per machine in such a setup. To describe a setup where batches vary depending on operation characteristics, as is the case for the sheet-metal production (Rinciog et al., 2020), we add $dbatch$ as a possible β value. It could be, that the batch content affects the batch processing time, as is the case with the aforementioned publication. In such a case we only know the operation duration p_{ji} post batch assignment. This additional complexity can be indicated by additionally placing dp_{ji} (dynamic processing times) in β.

In setups containing the no-wait parameter nwt, all operations need to be processed immediately after the predecessor operation has been processed. "For example, in aerospace product manufacturing, the inspection step must be immediately conducted after the heat treatment or aging treatment to ensure the quality of products. In steel manufacturing, the hot work-in-process must continuously go through the subsequent production steps to avoid unwanted cooling." (Lin et al., 2022) The partial no-wait β parameter $pnwt$ introduced by us describes situations where only a subset of the job operations need to be executed in a no-wait fashion. Such is the case in the paper by Lin et al. (2022).

Pinedo does not elaborate on the distinction between constraints of a stochastic nature and those of a deterministic nature. Release dates (r_j), could be deterministic, depending on the underlying planning process, but are most often stochastic (r_j^s) (e.g. Kuhnle et al., 2020). Similarly, breakdowns are deterministic ($brkdwn$), if resources are taken offline for planned maintenance, and stochastic in case of unexpected failure ($brkdwn^s$) (e.g. Hofmann et al., 2020). The presence of the demand parameter (dmd_j) implies that release dates are a system inherent choice. In a dmd_j system, finished jobs are consumed from a sink buffer as per incoming demand. A stochastic demand is indicated by dmd_j^s. This situation is considered by Mahadevan et al. (1998), Qu et al. (2015), and Paternina-Arboleda et al. (2005).

To round up the picture, we mention at this point, that there is a single further constraint defined by Pinedo that was not used in the surveyed RL literature, namely preemption – $prmp$. $prmp$ models situations where operations can be preempted, i.e. removed from a machine before having been completed (Pinedo, 2012).

The number of new β values defined and their frequency in the surveyed literature is a testimony to the need for the current work. As opposed to the α values, where only five of the 13 setups present in RL were defined by us, the β new parameters needed to be defined to accommodate all the setups in literature, the β parameters needed to be doubled to describe the studied setups. Of the four most popular parameters, namely stochastic release times r_j^s (24 publications), operation capabilities M_i^o (21 publications), stochastic resource failures $brkdwn^s$ (12 publications), p_{ji}^s (11 publications), the former three were

defined by us. Also, note the prevalence of stochastic parameters in the RL scheduling work. About half of the considered papers focus on stochastic setups.

2.2.3 Objectives

As can be seen from Figure 2.4, there are many possible optimization targets within the realm of production scheduling, depending on the emphasis on the particular use case. The green rectangles represent targets available delineated by Pinedo (2012), while the red ones were defined in the surveyed literature. Filled-in rectangles represent optimization targets used in the surveyed literature, while the ones not filled are defined in the aforementioned work but were not investigated by the RL literature. The gray rectangles represent intermediary variables needed for the target definition. The number of papers using the particular optimization target are noted down in parentheses within the blocks.

Central to nearly all objective function definitions are the job completion times C_j. To highlight its central importance, this intermediary variable is circled rather than boxed. Figure 2.4 contains them twice for spatial ordering reasons.

The zoo of optimization different optimization targets in literature is a clear statement of the diversity of production scheduling as a field. Because of this diversity, systematizing γ parameters is a daunting task, reflected by the apparent chaos in Figure 2.4. There is no such thing as a universal goal. Rather, the optimization goal is use-case dependent, and could potentially shift with time.

There are four main categories of optimization targets pertaining to how much gets produced (throughput, makespan), how fast jobs get done (flow-time), whether the production is timely (lateness, tardiness, unit cost – i.e. the number of delayed jobs), and whether the system is well utilized (machine utilization, vehicle utilization, job idle time, machine failures, buffer length, buffered times, setup times, inventory levels).

The most often encountered metric is the makespan, C_{max}, defined at the beginning of Section 2.2.1. While C_{max} makes sense for static contexts where there is a fixed number of jobs to be processed, for dynamic environments a throughput measure, could be more fitting since it makes scheduling problems comparable independent of the planning time horizon. In literature average job throughput $Tpt_{ave}^{j} := 1/t \sum 1_{\{C_j \leq t\}}$ at time t is used (Thomas et al., 2018; Kuhnle et al., 2020). Additionally, operation throughput $Tpt_{ave}^{o} := 1/t \sum_{j,i} p_{ji} \cdot 1_{\{s_{ji}+p_{ji} \leq t\}}$ could be considered. Flow-time (sometimes called lead time) F_j measures the time between job release and job completion, $F_j := C_j - r_j$. The average flow-time $F_{ave} := 1/n \sum F_j$ is an indicator of a system's reactivity/flexibility. Yet another measure based on the job completion time is the job idle time I_j defined as the difference between flow time and the summed job processing time: $F_j - \sum p_j i$.

The timeliness-related metrics are perhaps the most relevant for the industry (Schuh et al., 2013). Here the intermediary variable needed for all targets is the so-called lateness, i.e. the difference between completion time and due date, $L_j := C_j - d_j$. Note that lateness can be negative when jobs are finished before their due date. Tardiness extends lateness by

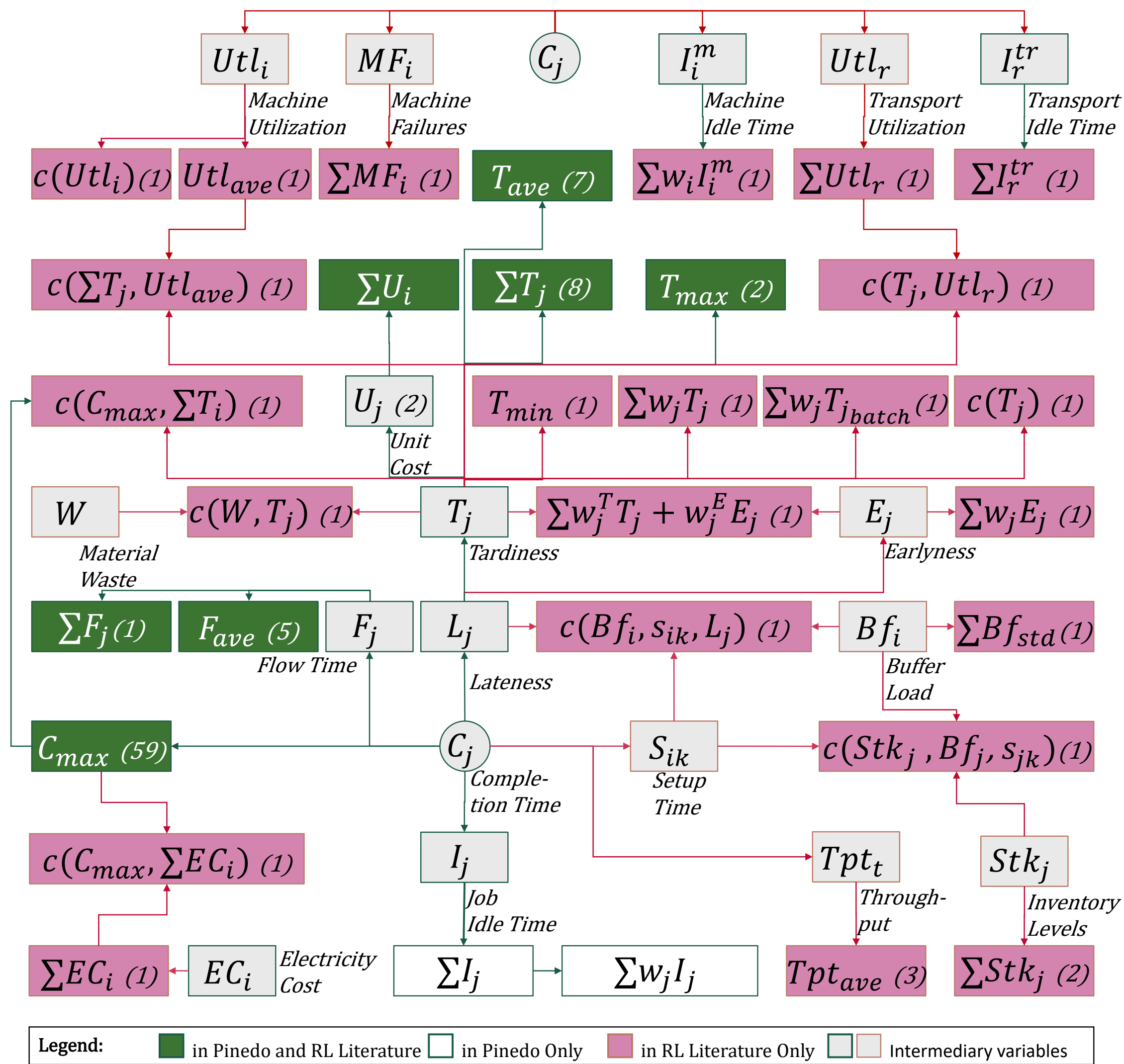


Figure 2.4: Optimization targets in the surveyed literature (γ). The number of publications employing particular targets are indicated in parentheses. Green blocks were defined by Pinedo (2012). Red blocks are defined by us. Gray blocks represent intermediary variables used in the target computation. White blocks are Optimization targets defined by Pinedo (2012) but were not encountered in RL literature. The job completion time intermediary variable C_j occurs twice for spatial ordering reasons.

ignoring early jobs $T_j := \max\{0, L_j\}$. The tardiness based optimization goals encountered are constructed by aggregating the tardiness variables, e.g. $T_{ave}, T_{max}, \sum T_j$. Alternatively, the number of tardy jobs ($\sum U_j$) can simply be counted using the unit cost variables $U_j := 1_{C_j > d_j}$ (e.g. Wang et al., 2005). Perhaps somewhat counterintuitively, finishing jobs too early can also be detrimental (Baker, 2014). As such, earliness $E_j := |\min L_j, 0|$ can be another minimization target as it is considered by Wang (2020) where tardiness and earliness are jointly minimized.

Resource utilization at time t could be defined as $Utl_i := 1/t \sum p_{ji} \cdot 1_{\{s_{ji} + p_{ji} \leq t\}}$, i.e. the time the machine was working over the total elapsed time t. Similarly one can define the transport utilization Utl_{tr} as the time spent carrying a load over the total elapsed time. The buffer composition can also be used as an optimization target. In general, one would like to keep buffer lengths to a minimum. In the publication by Qu et al. (2015) the number

of buffered operations Bf_i at resource i was used as an optimization goal within a more complex cost function also involving lateness and the total number of tool switches. While the related total processing time of buffered operations Bft_i at time t and resource i was only used as a state-space feature in the surveyed literature (Thomas et al., 2018; Zhou et al., 2020; Wang, 2020), it could indeed also be considered an optimization goal.

Complementary to utilization are the intermediary variables and related goals built around idle times. Both job idle times I_j and resource idle times I_i can be considered for minimization goal definition. With respect to resource idle time, we distinguish between machine idle times I_i^m and vehicle idle times I_r^{tr}. Goal examples for processing and vehicle idle times can be found in the works of Bouazza et al. (2017) and Kuhnle et al. (2020) respectively.

Outside of the traditional intermediary variables and optimization goals described above, other production aspects are sometimes considered for γ. Examples of such are the material waste (W) (Rinciog et al., 2020), inventory levels (Stk_j) for jobs j (Mahadevan et al., 1998; Paternina-Arboleda et al., 2005; Kuhnle et al., 2020), the number of failures (MF_i) of resource i (Mahadevan et al., 1998), and the electricity consumption of resource i (EC_i) studied by Wang et al. (2022) and Du et al. (2022).

These metrics listed thus far target different, possibly conflicting, aspects of the production system and can be combined into a joint optimization target (e.g. Paternina-Arboleda et al., 2005; Qu et al., 2015; Rinciog et al., 2020; Wang, 2020). This is done either by constructing a score as a function of multiple targets (scalarization), by simply measuring multiple scores individually, without involving *all* the measured metrics in the optimization target (e.g. Mahadevan et al., 1998; Chen et al., 2010; Kuhnle et al., 2019), or by following a Pareto front (e.g. Mendez-Hernandez et al., 2019).

A typical example of conflicting metrics is that of flow time and utilization/throughput as noted by Choo (2017). In dynamic setups this, i.e. those subsumed by dmd_j^s, a so called Work in Progress Window (WIP) is often used to control this trade-off (e.g. Hopp et al., 1991; Reyes et al., 2017; Barhebwa-Mushamuka et al., 2019). Additionally the WIP makes inventory management easier (Huang et al., 2008). The WIP limits the number of jobs to be scheduled concurrently to a fixed number, i.e. the WIP size. Jobs are added to the WIP when slots get freed as a result of WIP jobs being processed to completion. A large WIP size favors utilization/throughput at the expense of flow time. Note that the RL scheduling literature at hand either does not employ, or does not detail aspects related to the WIP. Also note that the WIP could be interpreted as a β parameter, should the scheduling setup impose its use. Alternatively, its use could be seen as a scheduling algorithm choice.

While many sui generis optimization goals are considered in the RL scheduling literature, the "traditional" targets are still the most frequent. Among them, the makespan $C_{\max}$ is by far the most popular with 59 of the 98 surveyed works using it as an optimization goal. It is followed by the tardiness-related goals present in 19 of the surveyed literature. Flow time related goals rank third being present in six of the 98 publications.

2.3 RL Modeling for Job Shops

In this section, we discuss how the authors of the surveyed literature designed and presented the RL models for the setups in the previous section. We introduce the MDP on a high-level in the next subsection and discuss several of MDP categories that succinctly describe the broad RL modeling approaches to scheduling. We call these categories MDP breakdowns and introduce them in Section 2.3.1. In Sections 2.3.2 and 2.3.3 we look at the state, action, and reward designs for RL-driven production scheduling and the employed agent algorithms respectively. Note that we do not go into details with respect to the employed algorithms at this point. Instead we refer to the original publications.

This section is supported by the information in Table A.2, which gives an overview of the modeling choices encountered. The columns are grouped into MDP characteristics and agent characteristics. The former category contains the MDP breakdown, along with the state- and action-space information, and whether multiple agents interact with each other within the same environment. We abstained from tabulating the reward-spaces encountered in literature, since there is no clear pattern that can be discerned herein. The agent characteristics are comprised of the class of RL algorithm (policy, value or both), the exact algorithm name and the representation of the agent function.

2.3.1 Markov Decision Process Breakdowns

RL is broadly defined through the interaction between a learner, or agent, and a so-called environment outside the agent as depicted in Figure 2.5. The agent senses the current environment-state and takes an action from a fixed set, whereby the environment is moved to a new state. This loop continues until an end-state is reached. Through a numerical reward signal that the environment provides, the agent receives feedback pertaining to his actions. The agent's goal is ultimately to take action in such a way, that the obtained reward is maximized. Through repeated environment interaction the agent's assessment of the actions maximizing future reward should improve.

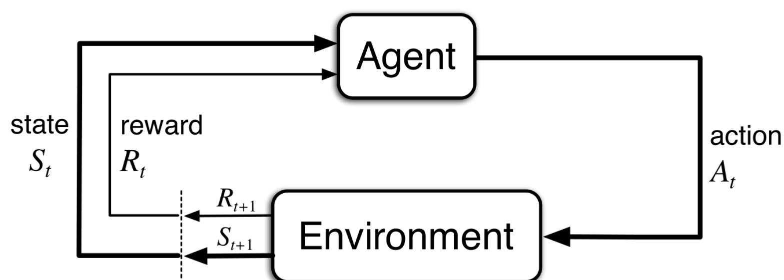


Figure 2.5: Abstract agent-environment interaction in RL: The environment presents the agent with a reward and a state. The agent takes an action that is fed back into the environment. A new state and reward signal is generated. (Source: Sutton et al., 2018)

The MDP describes the different elements of the interaction, lain down. Its components are the set of actions, or action-space, the set of states, or state-space, and the set of rewards, or reward-space, along with the environment logic, or state-transition function.

Note that there is a distinction between agent-state (view) and environment-state. While in the case of a "fully observable environment" the two are identical, within a "partially observable environment" the agent-state contains only a subset of the information in the environment-state.

To solve any problem with RL first an MDP must be modeled for the particular domain. While for many RL settings the design of agent-environment interaction is fairly obvious, for production scheduling there are many design choices to be made for each interaction component. Consider for instance the chess game for which currently an RL solution, namely AZ (Silver et al., 2016; Silver et al., 2017a) is state of the art. Here the interaction is quite obvious: the environment is a chess simulation, the state is a view of the board, an action is an eligible move, and the reward has one of three values depending on the win-lose-draw outcome.

Let us now consider $(FPOm|tr(a), r_j^s, brkdwn^s, p_{ji}^s|Utl_{ave})$. Here we have a considerably more complex simulation to code, and various design decisions to make. There are at least two types of decisions that should potentially be considered separately, namely job routing and sequencing. For both, there are several discrete events that could be used as a trigger for agent action, e.g. a machine finishes processing an operation or a vehicle has been freed, a job has finished, a new job has arrived, etc. Furthermore, the questions of action, state, and reward design still have to be answered.

Building an MDP for production scheduling boils down to first

1. breaking down the production scheduling process into discrete scheduling steps, then
2. choosing one or more RL agents
3. defining what exactly constitutes the relevant state information,
4. what a scheduling decision is, and
5. how to judge if an action is better than another by means of a reward.

Having written a simulation reflecting these modeling decisions the MDP is implicitly defined.

Our investigation of the pertinent literature reveals that the production scheduling process is broken down in one of the 11 ways, we call "MDP breakdowns". Our breakdowns can be can be grouped along three meta-categories. We describe the eleven breakdowns along with their meta-categories in the following, thereby contributing to a clearer communication of RL design choices.

Completely Reactive Breakdowns: Six of the 11 breakdowns are centered around either processing or vehicles becoming available and constitute "completely reactive scheduling" approaches (Bukkur et al., 2018). This means that no production plan is generated at any point in time. Instead, scheduling decisions are taken ad-hoc on particular cues (e.g. when a resource is freed).

The four main types of decisions that can arise on resource release are depicted in Figure 2.6. We use roman numerals to represent machine dependent decisions and letters to represent vehicle dependent decisions. When machines become available they require a new operation to work on (i). The eligible operations are the ones present in the machine input buffer. If the setup is additionally characterized by job routing flexibility, either because of the partial ordering of job operations (Om, POm), the machine replication (all the parallel setups), and/or because of machine (operation) capabilities (M_i, M_i^o), the downstream machine for the next operation from the job whose operation just finished needs to be selected (ii).

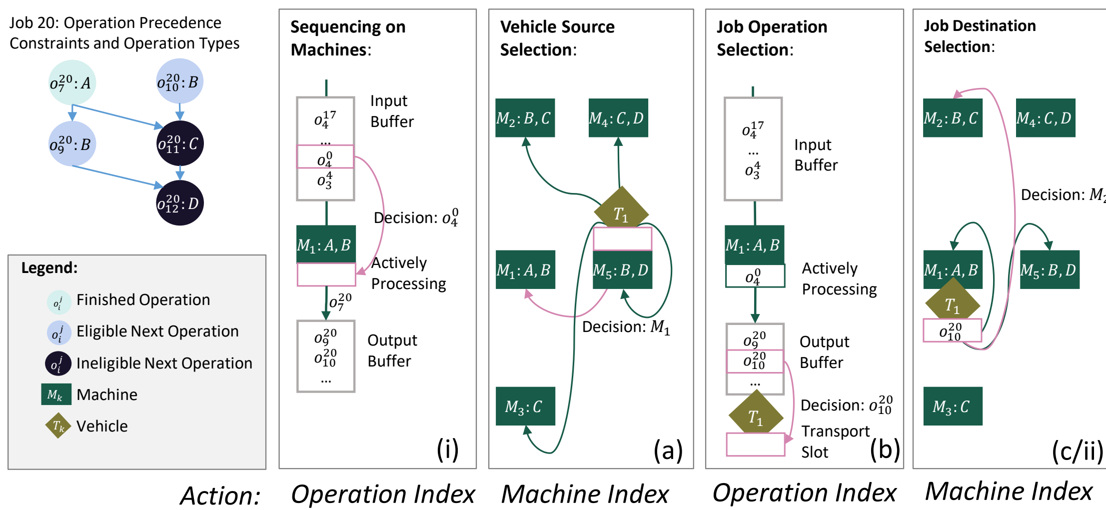


Figure 2.6: Overview of decisions within the context of purely-reactive scheduling.

Should vehicles be modeled explicitly ($tr(n)$, $tr(n)^s$, $tr(k,n)$, $tr(k,n)^s$ setups), two more decisions need to be taken into account. When a vehicle has become free by dropping off its load at a target machine, a new source destination needs to be selected for it (a). When vehicles reach a source destination, an operation from the source machine's output buffer needs to be selected for transport (b). Finally, after the operation is loaded, the vehicle destination, and therefore, implicitly, the job destination needs to be selected (c). Note that (c) corresponds to (ii).

We name the six completely reactive breakdowns *Iterative Sequencing*, *Routing Before Sequencing*, *Interlaced Routing and Sequencing*, *Transport-Centric Sequencing*, *Iterative Routing* and *Interlaced Tooling and Sequencing*. Bearing the decision types described above in mind, we can describe these breakdowns as follows:

1. Within the *Iterative Sequencing* breakdown, the moments when machines become free represent the discrete time steps when decisions can be taken. The free resource can then be assigned an operation from the ones queued in its input buffer (i). The operation would run on this machine for a duration specified by its processing time p_{ji} depending on the specification of the job j. After p_{ji} time units, the particular operation is marked as finished, the operations becoming eligible as a result are queued up in the input buffers of downstream resources and a new decision has to be made for the now free machine. This is both the simplest and the most widely spread

breakdown with 50 of the 98 publications considering it. In Jm and Fm problems, upon finishing an operation from a job, there is exactly one operation from the ones remaining in the job that becomes eligible for processing. Additionally, the eligible operation can be processed on exactly one machine downstream. As a result, the Iterative Sequencing is sufficient for defining all possible schedules. Examples of this can be found in most of the surveyed literature (e.g. Kim et al., 1998; Mahadevan et al., 1998; Riedmiller et al., 1999; Aydin et al., 2000; Gabel et al., 2007a).

Iterative Sequencing can be applied to environments with job routing flexibility as well. If the assumption of negligible transport times and endless capacity buffers is present, then all the downstream operations becoming eligible as a result of a particular operation being finished can be considered virtually queued in the input buffers of all their alternative resources concomitantly. The job routing choice (ii) is then defined implicitly through (i) by a machine picking one of these operations next. Naturally, the virtual operations that represented alternative job routes need to be removed from the respective input buffers.[6] Such approaches were investigated by Rinciog et al. (2020), Qu et al. (2015), Thomas et al. (2018), Waschneck et al. (2018), Zhou et al. (2020), and Luo (2020).

2. Within the *Routing Before Sequencing* breakdown, job routes are fixed in their entirety before the sequencing process. This is the approach taken by Martínez et al. (2011), who construct separate agents for job routing and sequencing. Whenever a new job arrives, the job routing agent iteratively assigns machines to it (ii), until it has a fixed completion path through production. Afterward, a sequencing loop starts where operations are pulled from the buffer when machines become available (i). Jiménez (2012) sets the job routes before sequencing for one of the three problem families considered, namely FJc. Note that this MDP could prove to be problematic since all the job routing decisions for a job are made before the production process has started, whereby the adaptivity of RL is partially relinquished.

3. *Interlaced Routing and Sequencing* is yet another option for dealing with the combined job routing and sequencing problem. Herein, the combined problem is solved in an interlaced fashion. Bouazza et al. (2017) decide upon the next machine (input buffer) to transport a job to on-demand when an operation is finished. This decision (ii) is then immediately followed by a sequencing decision (i) on the same freed resource.

4. *Transport-Centric Sequencing* is only encountered in setups with explicit vehicle modeling, e.g. $tr(n)$ (e.g. Kuhnle et al., 2020; Hu et al., 2020a), and reveals an additional way in which mixed job routing and sequencing decisions can be handled, namely through fixing one of the decisions to a heuristic. Within this breakdown, machines sequence operations following a priority rule, e.g. First In First Out (FIFO). Decisions are required when vehicles are idle. The agent alternates between the selection of source machines (a) and destination machines (c) The oldest job in the source output buffer is chosen (b) and transported to the destination input buffer.

[6]This assumes that job operations cannot be executed in parallel even if the job routing flexibility would allow it.

5. The *Iterative Routing* breakdown is analogous to Transport-Centric Sequencing in that it is encountered in setups with job routing flexibility and involves a fixed priority rule to take sequencing decisions. There is, however, a decisive difference between the two: In the case of Iterative Routing, vehicles are not modeled explicitly. In terms of Figure 3.1, decisions (a), (b) are not needed and decision of type (i) are fixed. Hence, the scheduling agent only needs to take job routing decisions (ii) whenever machines finish their current work. Such is the case with the FFc problem described by Jiménez (2012), and the FFc problem studied by Arviv et al. (2016).

6. The *Interlaced Tooling and Sequencing* breakdown is similar to Interlaced Routing and Sequencing, as the name suggests, with decisions being taken in an interlaced fashion. Different from Interlaced Routing and Sequencing, instead of job routing decisions, a tooling decision is made. As with most of the breakdowns before, decisions are required when resources finish processing an operation. All operations buffered at the resource have an associated toolset that needs to match the toolset of the machine in order for the operation to be executed on it. Within Interlaced Tooling and Sequencing, the resource tool set is first chosen by an agent. Then, an operation from the group of same-tool set operations is chosen for processing (i) as in Seq. Note that we did not explicitly model this decision type in Figure 2.6, as there was a single instance of this MDP sound in literature.

Planning Breakdowns: Reactive breakdowns have the advantage of being potentially more robust against unforeseen events, since with such modeling, a decision is always made based on the particular situation at time t. An alternative to this would be to create a plan a priori and simply execute it during production. The next four breakdowns follow this paradigm. To distinguish reactive scheduling from the ex-ante schedule creation, we use the term "planning" for the latter situation.

We named the four planning breakdowns encountered in literature are *Iterative Gantt Improvement*, *Iterative Edge-Definition*, *Direct Planning* and *Iterative Search Refinement*:

7. In the *Iterative Gantt Improvement* breakdown, there is no need for an event discrete simulation, as in the breakdowns before it. Instead, the state sequence of the RL loop consists of consecutive snapshots of production plans, which are represented as Gantt charts, where operations with their start and finish times are mapped to their respective resources. Individual snapshots can be either valid (e.g. Palombarini et al., 2019; Du et al., 2022) or invalid schedules (e.g. Zhang et al., 1995; Zhang et al., 1996). Given a particular plan, the agent selects between several operators that are then applied to the production plan seeking to improve or even repair it. Zhang et al. (1995) and Zhang et al. (1996), for example, first create a so-called "critical-path schedule", i.e. an idealized infeasible plan, by assuming endless resource capacity. The no-overlap constraint for machines[7] is then repaired by employing an RL agent to shift operation start times or reallocate resources.

[7]No more than one operation can be executed on any machine at any time. See scheduling definition in 2.2.1

8. Within the *Iterative Edge-Definition* RL planning paradigm (e.g. Zhang et al., 2020; Li et al., 2020; Seito et al., 2020) a plan is generated by using a disjunctive graph representation of the Jm setup and incrementally setting disjunctive edges. Decisions are being taken as long as the schedule is incomplete. The plan modified as per the agent's decision constitutes the new state and is fed back to the agent.

9. The *Direct Planning* approach is a fringe situation encountered in the works of Pan et al. (2021) and Wu et al. (2020). Herein the entire job sequence is fed into an Agent which then outputs the scheduling plan in its entirety. The plan is evaluated with respect to the optimization target, and a reward is generated for the agent. The process continues until the agent learning converges. The approach is tested on small ($Fm|perm|C_{\max}$) instances. While an interesting idea, it is doubtful that this MDP breakdown generalizes well to other setups. Since a schedule for $perm$ setups is described by a permutation over jobs, the action-space is kept small and intuitive. This is not possible for the vast majority of scheduling problems where production plans are organized as Gantt charts.

10. *Iterative Search Refinement* is a planning approach where RL plays a supporting role. Different iterative search algorithms represent the base planning method. The search algorithm can be anything from a local search to a complex heuristic (e.g. Min et al., 2022; Heger et al., 2021). Between search algorithm iterations, RL is used to manipulate the search parameters based on the current search state. Search state can refer to either search algorithm particularities, e.g. population features in the case of EA, the production state itself, or both. Cao et al. (2021), for example, set the step length control factor α during every search iteration of Cuckoo Search[8] (Yang et al., 2014) based on the current population features (i.e. the current search state). Similarly, Zhao et al. (2021a) use an RL agent to modify the Variable Neighborhood Search (Mladenović et al., 1997) parameters based on the current solution features.

Predictive-Reactive Breakdowns: An intermediary scheduling paradigm is the so called "predictive-reactive scheduling" (Bukkur et al., 2018). Herein an initially generated production schedule is adapted on occurrence of stochastic events. A single breakdown falling in this third category was encountered in the RL scheduling literature, namely *Iterative Re-Planning*:

11. *Iterative Re-Planning* can be used to describe the agent-environment interaction encountered in the work of Shahrabi et al. (2017). Here, the authors use a Variable Neighborhood Search approach which they parameterize depending on the particular production state. The events triggering a decision are the stochastic job releases. Upon the arrival of a new job, the RL agent chooses the parameters for the Variable Neighborhood Search, which is then used to formulate a fixed plan. This plan is followed until the next job release, and so on. ReP is closely related, but distinct from Iterative Search Refinement. As with Iterative Search Refinement, the RL agent parameterizes an external search algorithm. However, the parametrization happens

[8]Cuckoo Search represents a special class of EA, namely $\mu + \lambda$–evolution strategy (Villalón et al., 2021)

Table 2.1: Decisions included by reactive MDP breakdowns.

MDP Breakdown	Sequencing on Machines (i)	Vehicle Source Selection (a)	Job Operation Selection (b)	Job Destination Selection (c/ii)
Iterative Sequencing	X			(X) implicit
Routing Before Sequencing	X			(X) fixed ex-ante
Interlaced Routing and Sequencing	X			X
Transport-Centric Sequencing	(X) fixed rule	X	(X) fixed rule	X
Iterative Routing	(X) fixed rule			X
Interlaced Tooling and Sequencing	X + tool choice			(X) implicit
Holistic Routing and Sequencing	X	X	X	X

exactly once per plan computation and not over different iterations of search used to generate the same plan.

Discussion: The extension of the Transport-Centric Sequencing breakdown to accommodate sequencing decisions explicitly, so as to allow RL agents to access the scheduling solution space in its entirety, represents a new avenue of interesting research. We name this as of yet not investigated breakdown Holistic Routing and Sequencing. Table 2.1 provides an overview of the decisions associated with the different reactive scheduling breakdowns discussed thus far. Decisions taken by RL agents are marked by “X”. “(X)” marks decisions present in the scheduling system, albeit not necessarily taken by RL agent (e.g. fixed heuristic). The table shows that the proposed Holistic Routing and Sequencing MDP subsumes all other reactive breakdowns.

Figure 2.7 depicts the occurrence of the different breakdowns in the surveyed literature. The popularity of the deployed breakdowns both correlates with the breakdown simplicity and suggests a trend towards a mixed OR-RL approach to scheduling. The top four breakdowns in terms of their frequency in literature are Seq, Iterative Gantt Improvement, Iterative Routing and Iterative Search Refinement with 50 publications employing the first and the rest being considered by nine publications each. Iterative Sequencing and Iterative Routing are both relatively straightforward and cover 60% of the total publications together. Iterative Gantt Improvement is relatively easy to implement in terms of training and evaluation environment. Somewhat surprising is the popularity of Iterative Search Refinement, particularly in more recent years. This presents a high potential avenue of research combining the performance of traditional search approaches from the field of OR with the adaptability of RL methods. Note however, that Iterative Search Refinement is mainly tested on static setups. Also note that in about 13% of cases we were uncertain about the precise nature of the MDP breakdown at hand (“?” in the pie-chart). In some of

these cases we noted down both the most plausible category and the presence of inclarity and counted the paper both towards the hypothesized and the "?"-category.

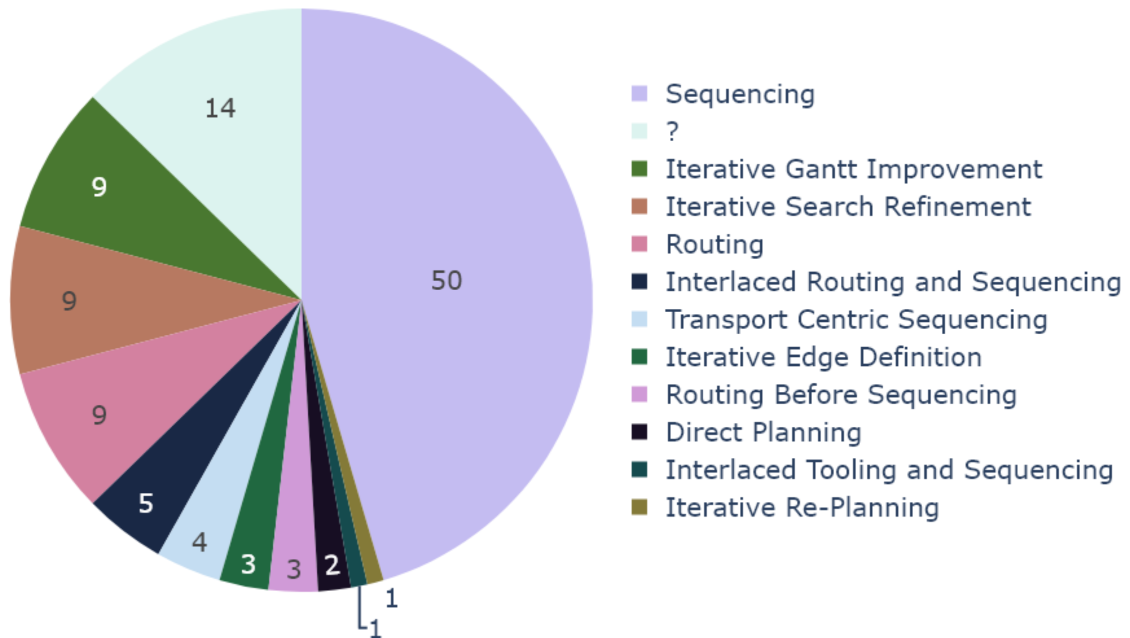


Figure 2.7: Overview of MDP breakdowns in RL scheduling literature.

Looking at the frequency of breakdowns in conjunction with the scheduling setups they were deployed to, reveals an additional gap, namely using RL to schedule setups with job routing flexibility dynamically. Save for Interlaced Routing and Sequencing, which is present in only five of the 98 papers (see Figure 2.7), none of the considered breakdowns solve mixed job routing and sequencing problems using RL without restricting the space of possible solutions. However, flexible setups are abundant among the surveyed papers. There are at least 34 publications tackling production setups offering job routing flexibility, judging by the frequency of the α variables alone (see Figure 2.1). Using approaches such as Iterative Routing or Iterative Sequencing for flexible environments (by fixing the complementary decision to a heuristic), for example, limits the space of possible schedules and may lead to the RL approach missing good solutions. Furthermore, in flexible environments, RL solutions may offer more advantages over traditional approaches, since the job routing flexibility adds an enormous overhead to search approaches by dramatically increasing the solution space.

From the presentation of the possible RL loops thus far, it should become obvious that there is no one way to solve a production scheduling problem with RL. To make matters worse, the exact formulation of the state-space ($\mathcal{S}$), action-space ($\mathcal{A}$), and reward signal (r) are also privy to considerable design choices, not to mention that there are many RL agents to choose from.

2.3.2 State, Action and Reward Modeling

States: The information based on which the state-transition occurs, also known as the environment-state, depends on the production setup considered and the MDP breakdown chosen. Take, for instance, a standard job-shop scheduling problem ($Jm \mid\mid C_{max}$) with an

Iterative Sequencing breakdown. Here, the complete state information could be encoded as four equal size matrices $T \in \mathbb{N}^{n \times m}$, $P \in \mathbb{N}^{n \times m}$, $L \in \{1, \dots, m\}^{n \times m}$, $A \in \{0, 1\}^{n \times m}$, the system time t and the index i of the machine requesting a sequencing decision. The matrix size is defined by the number of jobs n and number of machines m.

The type matrix T, keeps track of the precedence within jobs (rows) along with the type of resource $type(o_{ji})$ job operations o_{ji} need. The corresponding values in the P matrix keep tabs on the remaining processing times p_{ji} for each operation, with $p_{ji} = 0$ if the operation has finished. The location of all operations could be encoded using a matrix L with entries l_{ji}, where $l_{ji} = m_k$ if operation o_{ji} is in the input buffer or being processed at the machine m_k. To distinguish between operations being actively processed and those waiting in the machine input buffer, one could use the matrix A, with entries a_{ji} taking the values of either one or zero with one signaling active processing.

This is just one possibility of state encoding. One could, for instance, reduce the dimensionality of the state by making use of the precedence constraints within jobs. Since there is only one eligible operation per job at any time, L and A could be flattened to a vector. Yet another alternative would be to use two matrices P' and A' of dimension $o \times m$, where o is the sum of operations over all jobs, to encode the remaining processing time of all operations and whether they are being processed actively processed at time t or not. If all jobs have the same number m of operations and consecutive rows $mj, \dots, m(j+1) - 1$ are ordered as per operation precedence, this is all the environment-state information needed. Something similar was done by Waschneck et al. (2018).

If we present the environment-state information in its entirety to the agent together with a machine requesting a new operation, we are in a fully observable environment situation. This means that the agent has all the state-transition information the environment has and could, in theory, perfectly approximate the state-transition function p as well as reward function r.

Because of the sheer size of some of the scheduling problems, and/or their stochastic nature, this is not always possible. In fact, in all of the literature surveyed employing an Iterative Routing Iterative Sequencing or Iterative Gantt Improvement, save for a paper by Rinciog et al. (2020), full observability is not given. In the Iterative Edge Definition breakdown, full observability is given since the environment is not the production system, but rather its (incomplete) schedule (Zhang et al., 2020).

More often than not, only a part of the information used for the environment-state-transition is presented to the agents. This corresponds to a partially observable environment. Note that stochastic environments are only partially observable by design since the nature of the stochastic processes makes it impossible to perfectly foresee the next state.

Partial observability is also the norm in multi-agent Iterative Sequencing setups, where the agent only knows about the state information pertaining to the resource or job it controls (e.g. Riedmiller et al., 1999; Gabel et al., 2007a; Gabel, 2009; Jiménez, 2012). This is a design choice to improve scalability and ease the integration of new resources into an existing production system. If all the available local information is provided *local* full observability

ensues, which is not to be confused with full observability.

In Table A.2 we distinguish between two ways of constructing agent-states, namely **"raw"** information and **"features"**. By raw information, we mean that an agent receives an unprocessed view of the state directly from the simulation, such as one or more of the T, P, L, A, t, m_l information in the example above. Features imply that some environment-state information was condensed into some indicator(s). Note that in the case of Iterative Gantt Improvement, Direct Planning and Iterative Edge Definition the raw state is a partial scheduling plan, rather than a snapshot of shop-floor attributes. The partial plan of the Iterative Edge Definition state distinguishes itself from the other two by means of its representation, i.e. a disjunctive graph.

In terms of features, multitudinous options are available to the RL designer. Particularly older RL approaches relying on Q-value tables employ quantized features to keep the problem dimensionality in check (e.g. $Utl_{ave} < \tau_1, \tau_1 < Utl_{ave} < \tau_2, \tau_2 < Utl_{ave}$ for some thresholds τ_1, τ_2 defined by researcher). Such a quantization approach was taken by Shahrabi et al. (2017). Here, the states are represented as a discrete Cartesian product of job levels and total operation processing time levels.

Whether discretized or continuous, the employed features fall into one of three categories depending on whether they pertain to job properties (1), resource attributes (2), or an optimization target (3):

1. *Job-centric features* condense job information. Examples of such are inventory-backorder difference for all jobs (Paternina-Arboleda et al., 2005), remaining job operations, remaining job processing time (Gabel, 2009) or the number of jobs in the system (Shahrabi et al., 2017).

2. *Resource-centric features* target an aggregation of machine properties, such as remaining processing time in buffers (Zhou et al., 2020), resource workload (Gabel, 2009; Wang, 2020), the ratio of remaining processing times for machines and buffered processing time (Chen et al., 2010), number of product types in each buffer or machine health (Qu et al., 2015).

3. *Target-Centric features*, i.e. estimates of the target function, or of other targetable indicators are also often included in the construction of the state-space. Such examples are slack time (Aydin et al., 2000), estimated total tardiness (Wang et al., 2005; Luo, 2020), makespan estimate (Gabel, 2009), average machine utilization (Thomas et al., 2018; Luo, 2020), average vehicle utilization or the aforementioned average buffer length (Thomas et al., 2018).

Note that the first two categories are separated from the third by the granularity of the indicator. Average buffer length (e.g. Thomas et al., 2018), for instance, is an optimizable target in and of itself and falls in the third category. The vector with the buffer length of each individual machine (Wang et al., 2005; Aydin et al., 2000), on the other hand, falls in the second category.

Figure 2.8 shows the frequency of the different state modeling approaches. Extracting

features is the more popular technique, being used in 51 of the indexed publications (Zeng et al., 2011; Lin et al., 2019). Raw state information is presented to the agents in 38 of the 98 publications studied. Naturally, raw state information can be used in conjunction with features as is being done, for instance, by Thomas et al. (2018) and Kuhnle et al. (2020). The communication of state information suffers from a distinct lack of clarity with 20% of the publications presenting opaque or incomplete state descriptions (?). Note that the number of publications in the pie-chart is larger than the publications investigated. This is because we could often infer the rough agent-state category (raw or features) but not the precise state information. In such cases we noted down both the state category, and the lack of transparency (see Table A.2)

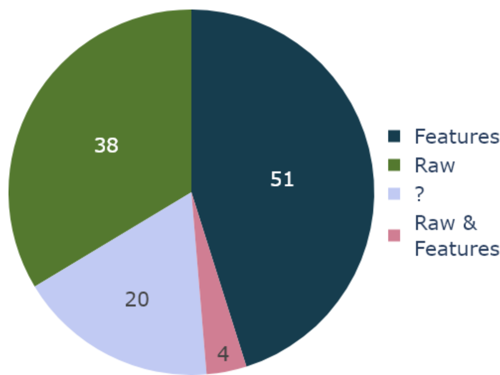


Figure 2.8: RL state modeling in RL scheduling literature.

Actions: In terms of actions, there are two main approaches to production scheduling. The agent either takes a **direct action**, thereby impacting the shop-floor directly, or outputs an **indirect action**, that is then used to select or parameterize an additional procedure, which, in turn, selects the direct action.

In the explored literature, direct actions are associated with one of the reactive MDP breakdowns. Direct actions are used the agent to directly select a target machine during the job routing process (e.g. Kuhnle et al., 2020) or a target operation for the sequencing step (e.g. Jiménez, 2012). When the precedence constraints limit the eligible next operations to exactly one per job (e.g. Jm, Fm), the job index suffices as an action, rather than the operation index (e.g. Gabel, 2009; Fonseca-Reyna et al., 2018). In the Interlaced Tooling and Sequencing breakdown, an index of a tool configuration from the tooling set can additionally be an action (Yang et al., 2021a).

The Direct Planning and Iterative Edge Definition scenarios force a different interpretation of direct actions. In the former breakdown, the entire schedule is created in one forward pass, hence the direct action is the schedule itself. In the latter breakdown, the action-space consists of the disjunctive edges of the schedule graph. The agent selects one of these edges to add to the graph. Since setting such an edge directly impacts the next state, it is considered a direct action.

The interpretation of direct actions is transparently dependent on the MDP breakdown. Indirect actions, on the other hand, are less transparent. As such, we use three different categories to make information more accessible, namely priority rules, Gantt operators and parameters:

1. *Priority rules* are a popular approach to modeling the agent actions for reactive MDP. The basic intuition is that different rules will be more effective in different production scenarios. Given a production state, the agent will output the index of a rule from a

fixed set. The rule is then applied to prioritize the operations in the machine buffer (sequencing decisions) or the viable downstream machines (job routing decisions). The highest priority operation/machine is then selected.

Mostly, the encountered priority rules are very simple. The most frequent examples of operation prioritization rules are Shortest Processing Time (SPT), LPT, Earliest Due Date (EDD), FIFO or Last In First Out (LIFO). The rule names are self-explanatory. Job routing priority rules function analogously. One could, for instance, prioritize the downstream machine with the shortest queue in terms of processing time, the one with the least elements, or the shortest setup times.

2. *Gantt operator* actions are characteristic of the Iterative Gantt Improvement breakdown. Within this breakdown, agents select an operator, e.g. a time-shift and reassignment operation (Zhang et al., 1996), which is then applied to the Gantt chart of the schedule to generate a new state. This indirect action class name reflects the operator domain and codomain, i.e. the Gantt representation of the schedule.

3. *Parameter* actions are present in the Iterative Search Refinement, Iterative Sequencing and Iterative Re-Planning breakdowns. The agent action is used to parameterize a single (planning) algorithm rather than choosing between multiple procedures (e.g. priority rules). Iterative Search Refinement examples include the publication of Cao et al. (2021) where different stages of a Cuckoo Search are parameterized, or Samsonov et al. (2021), where the agent selects a relative operation duration a, which is then chosen by a simple lookup algorithm to select the index of the eligible action with the duration closest to a.

   In the encountered Iterative Sequencing breakdowns, the parameter action given by the RL agent based on the current state is used to modify a sequencing priority rule (e.g. Heger et al., 2021; Min et al., 2022).

   The single Iterative Re-Planning approach found in RL scheduling literature was taken by Shahrabi et al. (2017). Here, the RL agent only gets deployed for action when a stochastic event, namely a new job arrival, occurs. At this point, a Variable Neighborhood Search is started to recompute a schedule for the operations not yet started. The RL agent action outputs the Variable Neighborhood Search parameters for the given state.

Figure 2.9 counts the different action modeling schemes present in the RL scheduling literature. Whenever the action-space was insufficiently described in the respective publication (17% of the publications), we added a "?" to the action-space column in Table A.2. We could often-times capture the action category despite the presence of inclarity, in which cases both the respective category and "?" were entered into the table. The most popular approach to action modeling is using direct-actions with 47 of the 98 indexed works employing this scheme. As a whole, indirect actions are almost equally popular, with 45% of the papers employing them. Among the indirect action schemes, the most popular is the priority rule approach (26% of publications).

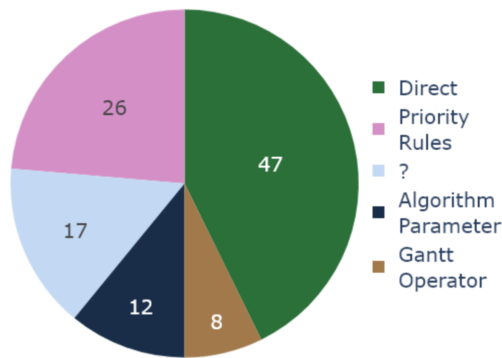


Figure 2.9: RL action modeling in RL scheduling literature.

Reward: The last modeling decision to be made pertains to the reward signal produced by the environment. In the investigated literature, the reward is most often proportional to the optimization target or a value that highly correlates with it (e.g. makespan and average utilization). That being said, there is no universally accepted scheme by which the studied authors construct their reward functions. Arviv et al. (2016), for example, who seek to optimize makespan, set the reward as a function of the minimum vehicle idle time and the minimum job waiting time. Gabel et al. (2007a), Gabel (2009), and Jiménez (2012) use completely different state information, namely the number of queued operations in the current machine's input buffer, to build the reward signal (the shorter the queue the higher the reward), though their optimization target is still the makespan.

Reward functions also differ based on when the agent receives a non-zero reward, for instance at every decision point, at the end of the (scheduling) game, or at any point in between. Other design choices include whether the reward is discrete or continuous (Matignon et al., 2006), strictly positive or both positive, and negative and many more (see Sutton et al., 2018). Finding an appropriate reward is of singular importance, given that it strongly determines learning convergence, and can be a challenging task for scheduling (Lang et al., 2020).

Since the reward signals vary wildly, we did not systematically take them into account, so as not to burst the frame of this work. However, during our experiments (Chapter 6) we investigated several of the rewards used in the scheduling literature to optimize makespan and tried out some of our own. We encourage the reader to jump ahead to the referred section for some concrete examples.

2.3.3 Agent Types

RL Algorithms: RL agents are trained according to the Generalized Policy Iteration principle (see Section 4.1). Herein two stages are distinguished, namely policy evaluation and policy improvement. During policy evaluation, agents select actions according to a policy function, and observe their returned reward. During policy improvement, the policy is adjusted based on the observations made. These stages are repeated until convergence (hopefully).

RL algorithms can be classified along three discrete axes (cmp. Sutton et al., 2018):

1. value, policy or actor-critic methods,
2. model-free or model-based methods, and
3. on- or off-policy methods.

In what follows we introduce these axes using algorithms encountered in RL production scheduling literature as examples. Note that we describe the last two categories for completion only, without tabulating the literature along these axes. This is because these two dimensions are somewhat more intuitive and less descriptive than the third.

Value-based methods try to estimate future reward by means of a value function, which is used to estimate the "goodness" of either states or actions from given states. During the policy evaluation stage, actions are selected by using the value function to ascertain the quality of the states reachable from the current one. The observed rewards are used during the subsequent stage to improve the value function. Examples of such methods are State Action Reward State Action (SARSA) (Rummery et al., 1994), QL (Watkins, 1989), DQN (Mnih et al., 2013), DDQN (Van Hasselt et al., 2016) and Dueling Dual Deep-Q Networks (DDDQN) (Wang et al., 2015) and Temporal Difference Learning (TD(λ)) (Sutton et al., 2018).

Policy-based methods are complementary to value-based methods. Within policy approaches to RL, agent policies π_θ are used directly to select actions during the evaluation stage. Based on the observed rewards, the reward expectation under policy π_θ is estimated, its gradient with respect to θ is computed and the policy is updated using stochastic gradient ascent. Examples include REINFORCE (Williams, 1987), and TRPO introduced by Schulman et al. (2015). Aside from these well-defined algorithms, in RL scheduling literature, we additionally encountered an unnamed policy-gradient method in Gabel et al., 2012.

Policy and value approaches can be combined into *actor-critic algorithms*. Instead of using environment interaction to approximate the expected reward directly, the (state) value function approximator (critic), is used to inform the policy approximator (actor) of the quality of its action. In RL scheduling literature, we find five different well-known actor-critic algorithms, namely Advantage Actor Critic (A2C) (Mnih et al., 2016), Asynchronous Advantage Actor Critic (A3C) (Mnih et al., 2016), Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015), PPO (Schulman et al., 2017), and AZ (Silver et al., 2017b). Note that AZ is not, strictly speaking, an actor-critic system, but rather a paradigm of its own. Nevertheless, it does use both a value and a policy function. Additionally, Han et al., 2021; Pan et al., 2021; Li et al., 2020 use what seems to be a Q-value actor-critic Value Actor Critic (VAC) (a precursor of A2C) without providing a name.

Figure 2.10 shows the frequency of value, policy and actor-critic approaches in RL scheduling literature. While the vast majority of approaches (70 of 98 papers) focus on value approximation, there is a growing body of work turning to actor-critic methods (17/98 papers). Pure policy approaches are relatively seldomly encountered (5/98 publications). The question mark indicates inclarity with respect to the algorithm class (5/98 papers).

On-policy methods (e.g. SARSA) use the same policy during evaluation stage that was adjusted during the improvement stage. This leads to more stable learning at the expense of exploration, which can lead to local optima. Conversely, in *off-policy methods* (e.g. QL,

DQN, DDPG, TD(λ)), the policy used during the evaluation stage can differ from the one used in the improvement stage, which leads to more exploration at the expense of convergence speed.

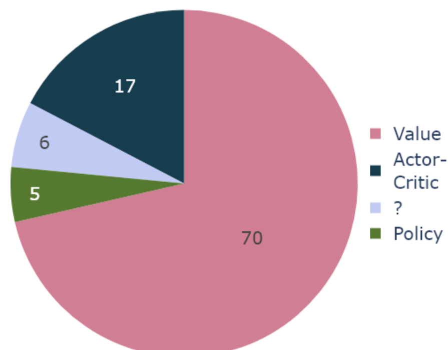


Figure 2.10: RL agent categories in RL scheduling literature.

RL algorithms can be further split into *model-free* and *model based* approaches. Save for AZ, all RL algorithms encountered in the production scheduling literature are model-free. Model-based approaches use an environment model to plan a few steps into the future before deciding on an action. The involved environment model is either estimated by the agent itself, e.g. Imagination Augmented Agent (Racanière et al., 2017), or simply given to it, e.g. AZ.

Figure 2.11 provides an overview of the frequency of different RL algorithms in the production scheduling literature. Notice the diversity of employed methods. QL, one of the most straightforward RL algorithms, is very popular with the RL production scheduling community, being deployed in nearly half of the surveyed works (44/98 papers). Its successor, DQN is the second most encountered approach (17/98 papers). When counting the DQN approaches, we do not distinguish between variations of the algorithm (e.g. DDQN). However, this information is present in Table A.2. Here too "?" marks inclarity pertaining to the employed algorithm.

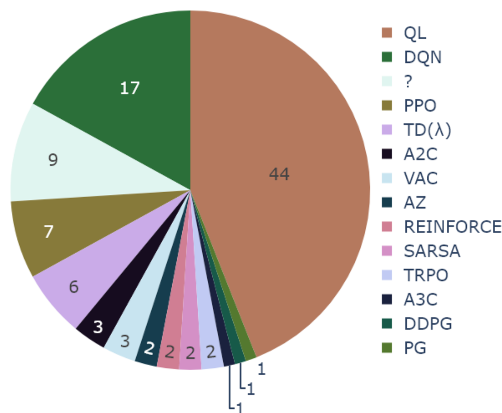


Figure 2.11: RL algorithms in RL scheduling literature.

The distribution of RL algorithms in the production scheduling literature hints at the fact that there should be more focus on approaches other than value-based. Particularly value-based approaches employing off-policy training with an Neural Network (NN) as function estimators, a combination also known as the "deadly triad", is problematic with respect to convergence according to Sutton et al. (2018). The deadly triangle (DQN, TD(λ)) occurs in 23% of the surveyed publications.

Function Representations: The data in the RL function column of Table A.2 and the counts of different RL functions in Figure 2.12 display a trend towards increasingly sophisticated neural approaches. While the tabular representation of RL (value) functions is still used by recent approaches, (e.g. Kim et al., 2022), the frequency with which it is encountered today has decreased. For tabular approaches (38 of 98 papers) to work, the state-space needs to be relatively small. The size requirement is met by leaving out information, by value quantization, or, as is done by Shiue et al. (2018), by employing the

SOM (Kohonen, 1990) clustering method to compress the function domain.

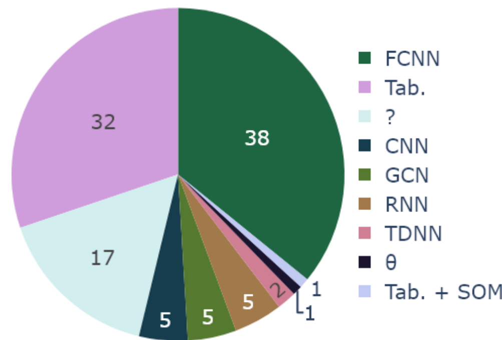


Figure 2.12: RL functions in RL scheduling literature.

Five different types of NN are used to map states to actions in as part of the agent function in RL production scheduling. The most popular type of employed NN (38 of the indexed publications) is the FCNN. The argument for it is the lack of spatial and temporal correlations between states (Rinciog et al., 2020). However, depending on the state modeling approach, these correlations may be present in the state, which is why Convolutional Neural Network (CNN) are used in four of the publications. The Time Delay Neural Network (TDNN) introduced by (Waibel et al., 1989) and used in RL scheduling literature by Zhang et al. (1995) and Zhang et al. (1996) can be seen as a precursor to CNN. Similarly, the modeling of the state as a graph structure within the context of the Iterative Edge Definition breakdown makes the application of GCN (Kipf et al., 2016) sensible. This is done in five of the surveyed publications. The NN thus far require fixed-size inputs. The five literature approaches employing Recurrent Neural Network (RNN) are capable of accommodating variable input sizes. Note that the function approximator need not be a NN. Zhang et al. (2013) use an analytic function parameterized by a weight vector θ instead.

Multi-Agents: The last RL agent aspect we briefly discuss is that of a multi-agent system. Rather than having a single decision entity, we can allow multiple RL agents to act within the same environment. These agents can be cooperative, i.e. striving to jointly maximize the expected reward or competitive, with each agent targeting a maximization of his reward only. These aspects, as well as further research-worthy multi-agent system modeling details, such as information sharing and goal definition exist (see for instance Jan't Hoen et al., 2005), are beyond the scope of this work.

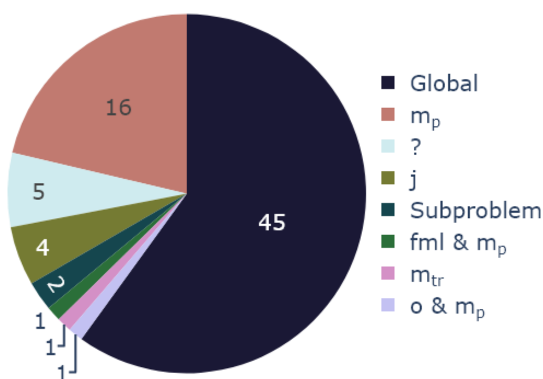


Figure 2.13: Agent deployment in RL scheduling literature.

Many of the authors in the surveyed literature employ such a multi-agent system (Figure 2.13). In all but one cases, within the Iterative Sequencing breakdown, the multi-agent system assigns one dedicated agent per machine m_p to decide the resources' operation sequence. A single exception to this rule can be found in the work of Paternina-Arboleda et al. (2005), where an agent is associated with each job and decides on the operation to follow. With respect to Iterative Routing, agents are deployed either per job j (Jiménez, 2012), operation (Martínez et al., 2011), job

family fml (Bouazza et al., 2017) or vehicle m_r (Arviv et al., 2016). The job routing agents decide on which machine to execute the next downstream operation on. Finally, agents can be deployed to solve a sub-problem. If the MDP contains different types of decisions, e.g. in the case of Interlaced Routing and Sequencing, machine indices for job routing and job indices for sequencing, dedicated agents are deployed for the different decision types, i.e. per sub-problem, as is the case with the publications by Yang et al. (2021a) and Wang et al. (2021a).

Note that authors sometimes describe systems where a single NN is used to make all decisions as multi-agent systems (e.g. Pol et al., 2021). The agents are said to "share the network", i.e. the policy, which is equivalent to a single RL agent being present in the system. *Multi-agent RL systems* are not to be confused with the much more general *agent-based modeling and simulation paradigm*. The former is mathematically well-defined within the frame of the Markov Game formalism. In particular, within Markov Games, each agent has its own individual policy (see Gronauer et al., 2022). The latter formalism encompasses the former and is considerably looser. Herein, agents are "any type of independent component" with behaviors ranging "from primitive [...] decision rules to complex adaptive intelligence" (Macal et al., 2005).

While, deserving of attention, multi-agent RL the apparent interest in such scheduling systems is somewhat surprising, given that single-agent schemes have yet to prove themselves within the domain. In particular, it stands to reason, that multi-agent RL scheduling systems should at least be benchmarked against single-agent approaches. We encountered a single instance of such an elaboration, namely in the work by Qu et al. (2015). Here, the authors benchmark a single agent against a multi-agent approach, with the single agent over-performing the alternative.

2.4 Validation

In this section, we take a closer look at what is needed in order to ensure reproducibility and provide sufficient evaluation of RL production scheduling experiments in Section 2.4.1 and Section 2.4.2 respectively.

The analysis at hand is based on Table A.3, which summarizes the surveyed literature in terms of five reproducibility and three evaluation criteria. The five reproducibility-related aspects we consider are, production setup clarity, RL design clarity, code availability, *simulation input availability* and reproducible stochasticity. In terms of evaluation we tabulate and discuss the train-test split, the state-of-the-art coverage through baseline algorithms, and, finally, the cherry-picking potential.

2.4.1 Reproducibility

Reproducibility is a condition sine qua non for ensuring sufficient validation. The following gives an example of how the lack of reproducibility can lead to insufficiently validated

approaches. Wang (2020) reports his Q-learning algorithm's[9] performance with respect to the results of the RL approaches reported by Lin (2009), Yang et al. (2009), and Wang et al. (2016) [10]. Of these authors, only Yang et al. (2009) benchmark the proposed method against something other than RL. The authors have shown, that the proposed method outperforms EDD, and SPT, but since experiments are not reproducible, the implicit comparison (by transitive property) between the RL approach and heuristics put forward by Wang (2020) does not hold and the paper falls short in terms of result validation.

Setup Clarity: By setup clarity, we mean the transparent delineation of the complete set of constraints required to recreate the studied setup along with the problem dimensions (number of jobs, number of machines machines etc.). The clarity requirement is a necessary albeit not sufficient condition for experiment reproducibility. In the presence of ambiguous problem and solution descriptions and the absence of code and simulation inputs, duplicating experiments can indeed become problematic, if at all possible. In Section 2.2 we saw how diverse the production setups considered by the RL literature are. In the absence of setup description standardization, it becomes difficult not to forget to include all production setup details.

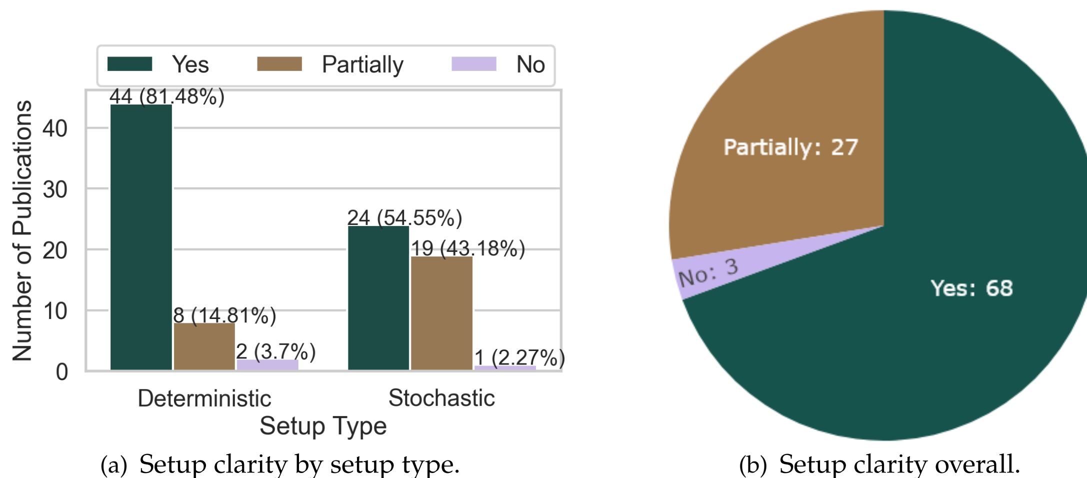
From Figure 2.14, we see that in the literature at hand, the presented setups were mostly described transparently (in 68 of 98 publications). The figure treats deterministic and stochastic setups separately. The percentages marked in parentheses are relative to the deterministic and stochastic subcategories respectively. The publications in the deterministic category, of which 81% are clear, fare much better than the publications dealing with stochastic setups, where only 54% are sufficiently detailed. This is not surprising, given that the encountered deterministic setups tend to be simpler and rely on available benchmarking instances. If only details related to the problem size were missed, we consider the respective publication to be partially clear. In any core constraints defining the problem itself are missed (e.g. operation precedence within jobs) then, we count the work as unclear.

Overall, most of the setup constraints are presented clearly save for the exact problem size (e.g. number of machines in each work center) and sometimes some constraint parameters (e.g. buffer capacities, number of transporters, machine-to-operation type mapping). Particularly stochastic setups are victims of such omission (e.g. Stricker et al., 2018; Waschneck et al., 2018; Kuhnle et al., 2020). Moreover, for setups including dynamic job release dates, the number of jobs considered in total during the simulation execution is sometimes in-transparent.

The total number of jobs tested should always be provided, as was done for example by Chen et al. (2010) and Luo (2020). Providing the experiment length in terms of total executed operations (e.g. Shahrabi et al., 2017), is insufficient, because the problem complexity is thereby obfuscated. Take for instance a Jm setting with 100 operations. The

[9]WQ_CDS stands for "Weighted Q-learning [...] based on clustering and dynamic search" (Wang, 2020); the algorithm seems to be a variation of QL using a clustering technique to control the state space dimensionality.

[10]All the algorithms are variations of tabular Q-learning; "B–Q learning algorithm [with an] adaptive scheduling control policy (BQ_ASCP)"(cmp. Yang et al., 2009); "state membership grade weighted Q-learning ([...]SMGWQ)" (cmp. Wang et al., 2016)



(a) Setup clarity by setup type.

(b) Setup clarity overall.

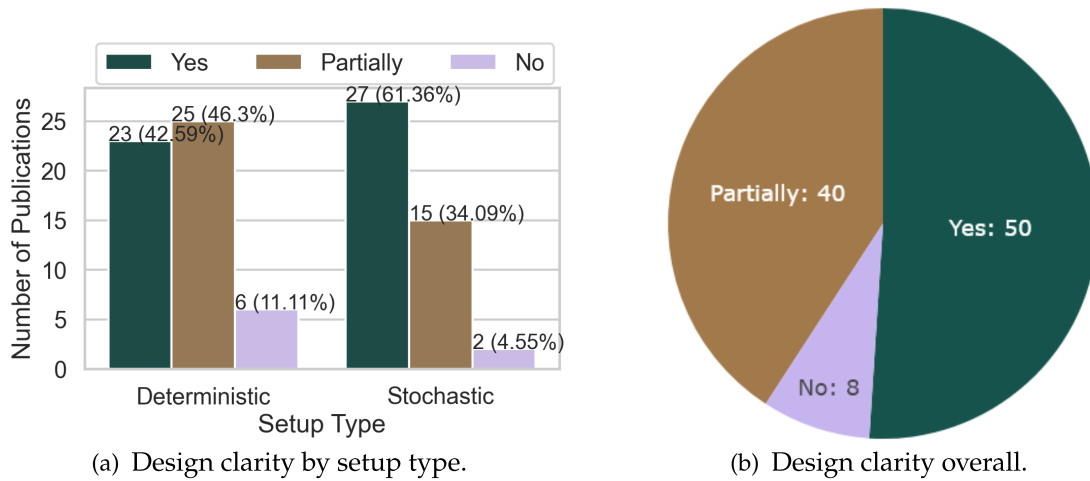
Figure 2.14: Setup clarity in RL literature.

combinatorial complexity of two jobs of length 50 is considerably lower than that of ten jobs á ten operations. Using the number of decisions taken by the agent (Jiménez, 2012; Qu et al., 2015; Stricker et al., 2018; Kuhnle et al., 2020), or, equivalently, the number of time-steps (Paternina-Arboleda et al., 2005) in dynamic setups is not particularly useful, since the agent behavior can, depending on the simulation implementation, influence the total number of jobs that passed through the system. Consider, for instance, implementations where agents can output a wait signal. It is also important to note, that RL algorithms explore their environment in a stochastic fashion either because of the stochastic nature of the function approximators (e.g. NN initialization) or because of the selected exploration-policy (e.g. ϵ-greedy). Because of this one cannot guarantee the same experiment execution a second time.

RL Design Clarity: The design clarity indicator considers whether the MDP components associated with a particular experiment are sufficiently detailed to allow for their re-implementation even in the absence of code. Because of all the design choices involved in deploying RL methods to production scheduling, the MDP breakdown, state- and action-space, reward function, and agent algorithm may not always include all the details required to emulate experiments. We distinguish between situations in which some but not all of the MDP elements are clearly delineated (partial clarity), and situations in which the none of the elements are clear, or the RL loop itself, i.e. the MDP breakdown, is unclear (inclarity).

Figure 2.15 displays the RL clarity aspect of the considered publications. Form it, we can see that almost half of the papers (48 out of 98) contain at least some elements which would impede the exact recreation of the lain down designs. Overall, publications dealing with stochastic setups tend to fare better than papers on deterministic setups, with 61% of the former and 41% of the latter being sufficiently clear.

More effort should be put into a clear presentation of the RL design elements. Particularly the state and action designs suffer from a lack of clarity in the surveyed literature. Sometimes the lack of clarity gains a transitive dimension. Reyna et al. (2015), for instance,



(a) Design clarity by setup type.

(b) Design clarity overall.

Figure 2.15: RL design clarity in RL scheduling literature.

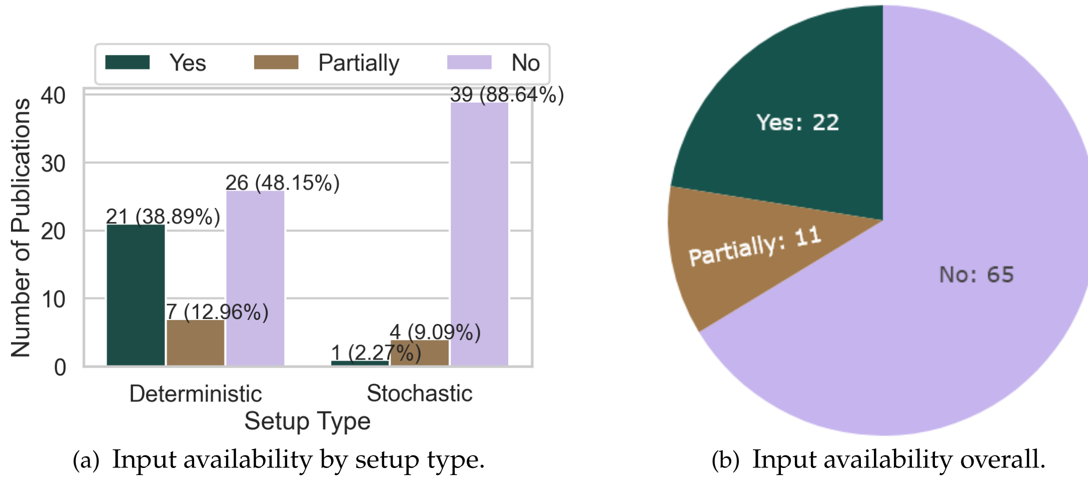
model the state-space as per Gabel et al. (2007a), where the state-space information offered to the agents is murky. As such, the reader is still somewhat in the dark. Additionally, when using NN as function approximators, there is a need for an exact account of the network architecture employed, as well as its parameters.

Input Availability: In order to compare production scheduling approaches, the exact inputs, i.e. the job sequence including the operation sequence within jobs and the operation duration, need to be provided. In the case of standard ($Jm \mid\mid C_{max}$), ($Om \mid\mid C_{max}$) or ($Fm \mid\mid C_{max}$), such inputs are available in the OR library and other repositories. These enable authors to compare diverse scheduling approaches by simply using the reported target makespan in previous research for those particular instances. Most of the deterministic setups use these inputs and corresponding previous results for benchmarking. Much of the literature surveyed, however, expands the production settings considered to more closely match the production reality.

Figure 2.16 gives an overview of the number of publications satisfying the input availability criterion. 65 out of 98 publications do not satisfy this requirement. Partial satisfaction of the input availability requirement (11 out of 98 papers) means that the inputs are based on some benchmark, albeit extended, with the extended inputs not being provided exactly (e.g. Brammer et al., 2022). The input availability problem is significantly more acute for stochastic setups. Here a whopping 88.64% of the publications are found lacking.

In stochastic cases, the illusion of input availability is creating through the provision of the sampling scheme used. Note, however, that this is often not sufficient for problems with a high degree of variance, which is characteristic of most production scheduling problems. Running experiments on randomly sampled inputs has the caveat of requiring a large number of experiments and statistical testing to ensure comparability with other approaches in the absence of reproducibility. Such an explicit statistical analysis is absent from the RL scheduling literature.

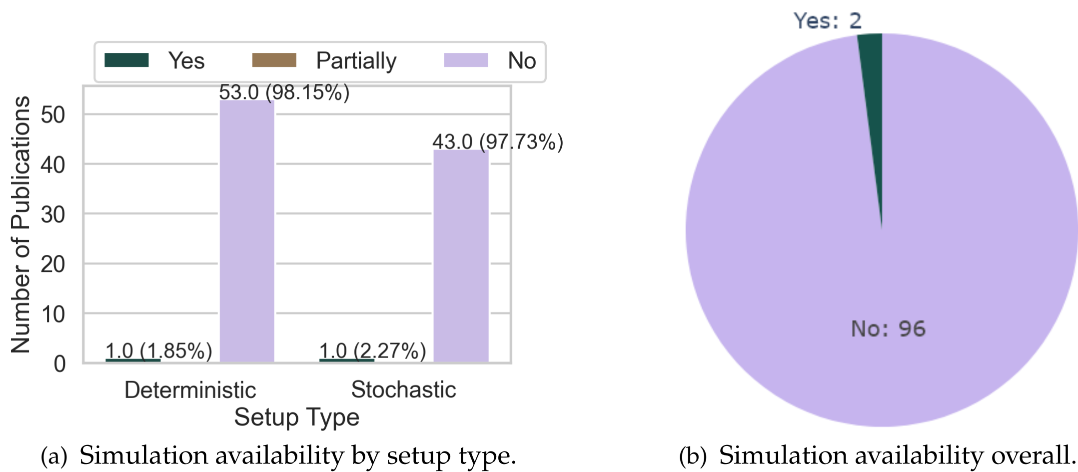
Code Availability: All but two of surveyed works fail to provide the associated simulation code (Figure 2.17). For a deterministic setting with a clear formulation and transparent



(a) Input availability by setup type. (b) Input availability overall.

Figure 2.16: Input availability in the surveyed literature.

inputs, the absence of simulation code, while leading to redundant implementation work, does not impact the usefulness of the study's results, since future work can still compare against the results reported. The scientific community as well as the industry would still profit from published code, as this would free up energy for more complex problems.



(a) Simulation availability by setup type. (b) Simulation availability overall.

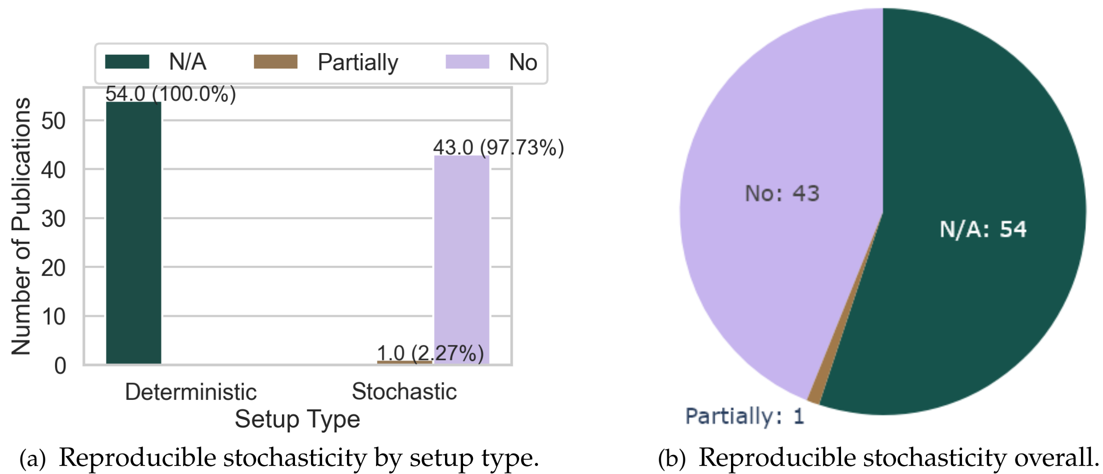
Figure 2.17: Simulation availability in the surveyed literature.

Reproducible stochasticity: Code availability alone does not suffice to ensure reproducible stochasticity, though it is a necessary condition in the absence of input availability. There is a simple way to ensure that a sequence of sampling actions produces the same output irrespective of the system the simulation is run on: Random Number Generator (RNG) seeding, as noted by Kuhnle et al. (2020). While here the authors do employ RNG seeding, they note that two runs with different controls will not necessarily end up with the same sequence of jobs, since different controls can lead to different processing sequences, which in turn may influence the meaning of a new sampling action, e.g. the nonce could be used to generate a new job or decide whether a machine fails.

To ensure that the occurrence time of the stochastic events is independent of the scheduling algorithm, one should sample *all* the stochastic events along with their occurrence time

during simulation initialization. These events can then be queued and triggered at the appropriate time. In doing so, it can be ensured that the same nonce is always used in the same way. A simulation implementing this scheme is described in Chapter 3. An added benefit of this scheme, is that the instantiation of all stochastic variables can be easily saved, thus enabling the satisfaction of the input availability requirement.

Figure 2.18, gives an overview of the publications meeting the reproducible stochasticity criterion. The figure demonstrates that this is, indeed a significant gap in the RL scheduling literature: With one exception, Stochasticity is not reproducible for any 44 stochastic setups reviewed. We deemed the stochasticity present in the work of Kuhnle et al. (2020) partially reproducible, given the author's own comment of different controls leading to potentially different stochastic events. For deterministic setups (54 papers), this condition is, of course, not applicable (N/A).



(a) Reproducible stochasticity by setup type. (b) Reproducible stochasticity overall.

Figure 2.18: Reproducible stochasticity in RL scheduling literature.

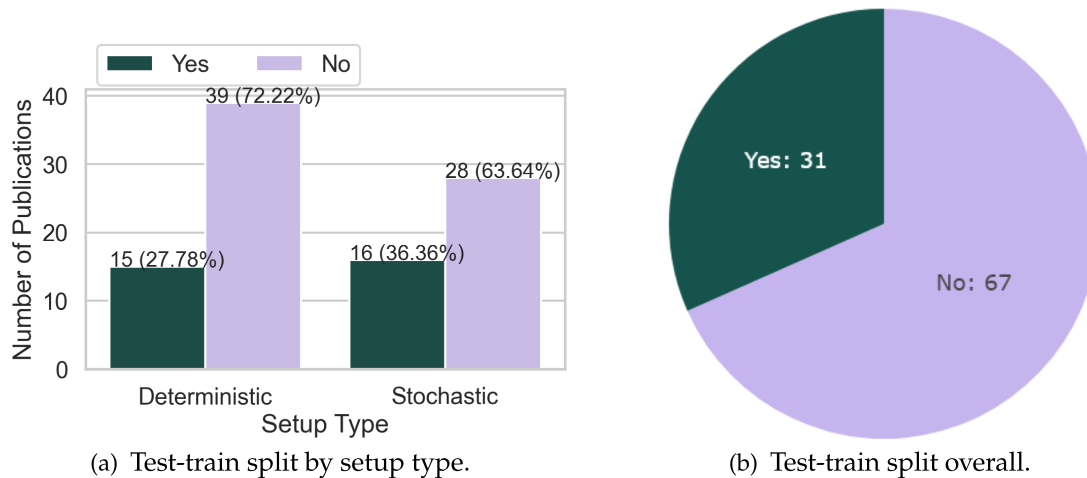
2.4.2 Evaluation

The last three criteria we used to order the reviewed literature pertain to how the RL approaches are being validated against preexisting solutions. First, we consider whether there is a separation between production instances used for training and those used to report the results on, i.e. a test-train split. Secondly, we checked whether the RL approaches are benchmarked against both established search approaches from the field of OR and heuristics. Finally, we loosely assessed the potential for voluntary or involuntary cherry-picking.

Test-train Split: As with all supervised or semi-supervised ML approaches, the phenomenon of overfitting can occur for RL schedulers. This refers to a situation where a strategy is developed that is very well suited for the training instances but cannot generalize well to an unseen situation (test instances).

Figure 2.19 displays the ratio between the publications employing the test-train evaluation technique and those that do not while distinguishing between deterministic and stochastic setups. Altogether, a total of 31 authors employ the technique, while the other two thirds

do not. 39 of the 54 deterministic setups in the reviewed papers (72%) do not make use of the test-train split. In stochastic cases, a lower percentage of publications (63%) of authors report their results on the training set only.



(a) Test-train split by setup type. (b) Test-train split overall.

Figure 2.19: Test-train split in RL scheduling literature.

Not differentiating between test and training set in the case of deterministic production (e.g. Zhang et al., 2013; Wang et al., 2021b; Samsonov et al., 2021) is equivalent to assuming that the scheduling instance to be optimized is recurrent. In such a situations RL looses its attractiveness: The speed argument evaporates, since *RL training times are quite extensive*, and the adaptivity argument is not pertinent convincing given the absence of stochasticity. In such cases, one may be better served by using other, more established search approaches, such as EA, local search, or even exhaustive search if the instance is small enough. This is reflected by the literature, seen as whenever the comparison is available, RL fails to outperform the literature's lower bounds for makespan.

Conversely, the presence of a train-test split for deterministic problems (e.g. Gabel, 2009; Park et al., 2021; Ren et al., 2021b) suggests a situation which is (potentially) better suited for RL. While the training process is slow, during *deployment, RL algorithms are fast*. If an RL agent can generalize well, it could be first trained with no time-constraints on some instances and be then deployed to different, unknown instances where time is of the essence. In such a non-recurrent situations, if the size of the problem is considerable, RL may indeed be a preferable solution as compared to planning approaches with long convergence times, e.g. EA.

In the dynamic setting, the test-train separation is not mandatory, since the stream of product specifications induces a test-train split, with the beginning of the stream being used for training and the latter part for testing. More generally, the train-test split is also not mandatory in other stochastic situations even if the problem is static, e.g. those with stochastic processing times, machine failure, etc. Given the stochastic nature of the respective instances, though the job structure and stochastic event distribution(s) may be the same, the occurence of stochastic events still varies. Here the agent's generalization capabilities are still tested within the same scheduling instance. Note however, that,

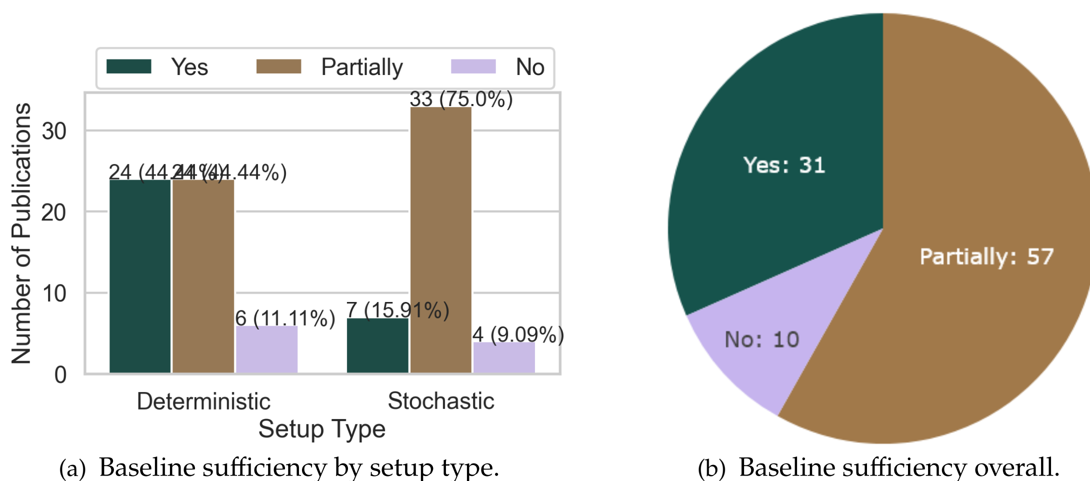
depending on the amount of stochasticity present in the system, re-planning approaches, e.g. using exact methods, may very well still outperform RL.

While not mandatory, some authors (e.g. Kuhnle et al., 2020; Luo et al., 2021a) still employ the split in such cases to additionally test the transfer learning capacity of the agents. The difference between generalization and transfer learning is mainly given by the distance between training and testing domains. Training a model to recognize synthetically generated object images and testing it on the real-world images, for example, constitutes a transfer learning task. In scheduling terms, transfer learning is investigated when the agents are trained and tested on significantly different problem instances (e.g. different problem size, different stochastic event distributions), or even different problems altogether (e.g. training of FJC and deployment to Jm).

Sufficient Baselines: Sufficient baselining entails pitting RL against both simple heuristic approaches *and* more established search techniques from the field of OR, e.g. EA, CP. If only simple heuristics are used for comparison, we note down a partial baseline sufficiency for the respective paper.

For static production settings most approaches are sufficiently validated by comparison with either the optimal results or the state-of-the-art search approach. However RL solutions for stochastic settings are almost exclusively baselined against heuristic solutions. It could very well be, however, that constructing schedules as if there were no stochasticity and simply ignoring plan deviations thereafter could be a competitive approach to RL or heuristics. Yet another approach would be to recompute a schedule on occurrence of unforeseen events, as is done by Shahrabi et al. (2017).

Figure 2.20 points towards a baselining gap for RL scheduling approaches. 67 of the explored works benchmark their approaches insufficiently. Of those, ten do not provide any viable baseline at all. Note that most of the correctly baselined approaches are tested on deterministic setups (24 papers). This makes up 44% of the category, which is still surprisingly low. In the stochastic case, the baselining gap is even wider, with only 16% of experiments in this category being compared with both heuristics and search.



(a) Baseline sufficiency by setup type.

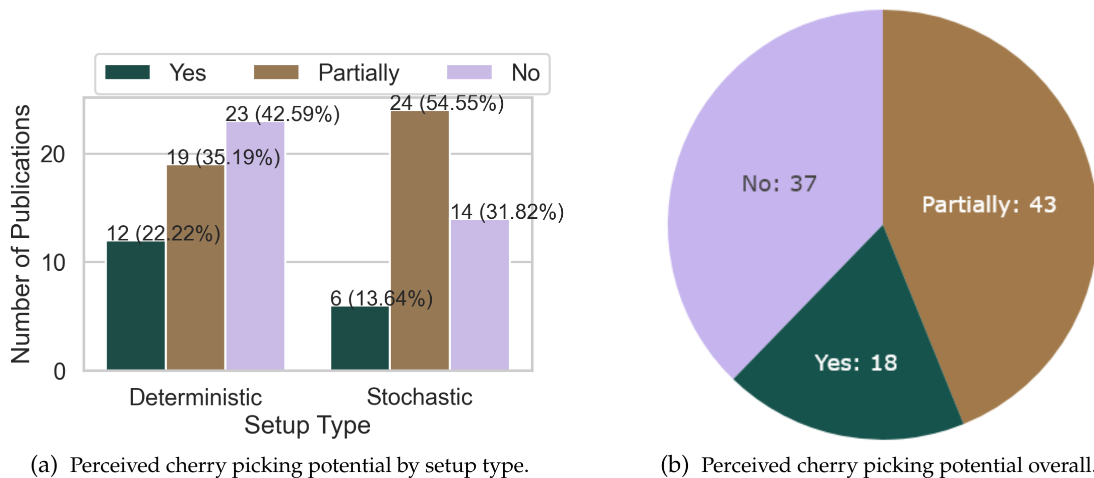
(b) Baseline sufficiency overall.

Figure 2.20: Baseline sufficiency.

While finding optimal solutions is not tractable for large instances, proof of concept experiments can leisurely be performed for smaller instances by using a CP or MILP solvers such as the ones offered by the ORTools[11] library or those available through the MiniZinc[12] integrated development environment. Such solvers often allow for the use of a time limit for the solution space search procedure. Therefore, RL solutions for large production instances could at least be benchmarked against search, which is a standard OR strategy.

Cherry-Picking Potential: Cherry picking as a concept is very simple: One runs n experiments, but only reports the results on $m < n$ of those that support the hypothesis. Whenever the results are reported on only a small number of experiments, the experiment design is not reproducible, and there is no statistical testing involved, this lingering suspicion remains. Most surveyed publications concerning stochastic setups suffer from this caveat, which is sometimes noted in the literature as well (e.g. Cunha et al., 2021). To avoid cherry-picking suspicion, a sufficiently large number of experiments should be run and statistical significance testing should accompany the scheduling results.

Figure 2.21 visualizes the distribution of the perceived cherry picking potential in the reviewed literature. In 18 publications, just a handful of experiments are ran on stochastic problems with a high degree of variance. In this cases the potential for cherry picking is quite high. Most publications (43) are in an interim stage with at most hundreds of experiments conducted. In 37 publications thousands of experiments are ran or the statistical significance of the described comparisons is explicitly analyzed.



(a) Perceived cherry picking potential by setup type. (b) Perceived cherry picking potential overall.

Figure 2.21: Perceived cherry picking potential.

2.5 Research Gaps

From the present standardization effort, several significant research gaps become apparent. The missing pieces can be grouped along the following six dimensions.

Standardization Gap: The standard introduced here should be maintained, extended,

[11] https://developers.google.com/optimization
[12] https://www.minizinc.org/

and implemented. The tables in this work should be maintained and extended as needed as new work surfaces in the field. Additionally, the information used in the definition of states, actions, and rewards should be exactly extracted and formalized. For instance, we should define individual features and rewards and not just their categories, as was done here.

Benchmarking Simulation Gap: The present standard should be used to implement or extend an open-source benchmarking simulation that ensures experiment reproducibility. Such a simulation should target the more general production scheduling problems in terms of base setups and additional constraints and allow for the monitoring of varied scheduling targets. Furthermore, because of the many options for RL design, it should offer researchers the possibility of configuring the associated state, action and reward-spaces and be compatible with available RL agent libraries, such that the agent implementation overhead is additionally eliminated. The simulation should be extensible to allow for the embedding of new constraints. Finally, the simulation should be efficient in terms of runtime, since RL algorithms are immensely data hungry.

Comperitive Baseline Gap: A set of competitive benchmarking algorithms should be developed for the stochastic cases. The benchmarking algorithms should share some of the traits that make RL attractive, e.g. they should be fast and adaptive.

Stochastic Experiments Gap: More RL experiments focusing on stochastic setups should be conducted. Intuitively, RL agents can be trained to adapt to unforeseen situations, while still using the expectation of future rewards to plan a little. As such, they position themselves between myopic heuristic approaches and the re-planning approaches which could be easily broken by stochasticity. RL has been deployed for solving both deterministic and stochastic production scheduling problems with varying degrees of success. In standard deterministic setups, RL generally fails to outperform the state of the art (e.g. Gabel et al., 2012; Reyna et al., 2015; Arviv et al., 2016; Fonseca-Reyna et al., 2018; Mendez-Hernandez et al., 2019; Zhang et al., 2020). For stochastic setups (e.g. Hofmann et al., 2020; Hu et al., 2020b; Liu et al., 2020; Luo, 2020; Kuhnle et al., 2020) and highly complex static setups (e.g. Zhang et al., 1996; Rinciog et al., 2020) RL shows more promise. We theorize, that whether re-planning, RL, or heuristics work best, is highly dependent on the amount of stochasticity inherent to the problem considered.

Stochastic Benchmarking Setup Gap: Seen as the burden of proof still lies with the RL scheduling field to showcase the situations where RL is a better candidate than its alternatives, experiments should focus more on setup details. In particular, more effort is required with respect to the quantification of the level of uncertainty inherent to particular instances. Benchmarking datasets for the different problems and different stochasticity levels should be constructed using a simulation guaranteeing reproducibility. This would allow for the comparison of the many RL design options available on the *same* problem, such that a more informed choice can be made pertaining to the best models.

RL Design Gap: There is a need for a transparent discussion of the model selection process not only in terms of RL algorithm parameters, but also in terms of the the available design

options. Because of the many design and parameter choices available for MDP definition, and the RL algorithms themselves, training (deep) RL solutions for production scheduling is an uphill battle. While MDP for board games such as chess or go (Silver et al., 2016), or computer games such as Atari (Mnih et al., 2015) are fairly straightforward to design, production scheduling is the opposite. Many design options are available for the MDP components in terms of actions, states and rewards.

Chapter 3

FabricatioRL: A Benchmarking Simulation Framework for RL Production Scheduling

When you design without simulation you end up overdesigning
— Justing Hendricksen, ANSYS Inc

Although studies have shown RL to be promising, there is a validation gap calling for a simulation framework enabling researchers to robustly embed RL methods within the state of the art (see 2.5). This can be best achieved by ensuring that the simulation allows the reproduction of the scheduling problem's inputs as well as other stochastic effects by means of variance control techniques.

A well designed benchmarking simulation would reduce the overhead associated with RL scheduling experiments significantly. First, the work associated with building the scheduling setup would be eliminated or at least reduced, depending on whether the simulation needs to be extended or not.

Secondly, a simulation respecting RL standards would help increase the diversity of RL methods employed for scheduling. Production scheduling literature mainly employs variations of QL or DQN as scheduling agents (see Table 2.11). While popular and easy to implement, QL, and DQN are by no means the only options. A simulation framework compatible with RL libraries such as KerasRL (Plappert, 2016), Stable-Baselines (Hill et al., 2018), Stable-Baselines3 (Raffin et al., 2021), Horizon (Gauci et al., 2018) or Tensorforce (Kühnle et al., 2017) would make the deployment of varied RL algorithms to production scheduling problems much easier. The less time spent implementing RL algorithms could also lead to a focus shift towards experimenting with various RL designs.

Thirdly, the validation effort for stochastic setups would decrease. In the absence of reproducibility, researchers seeking to compare the results of a new scheduling method to results reported in literature need to re-implement the methods described by the authors

and re-run the reported experiments. However, if variance control is implemented correctly, authors can simply reproduce the scheduling instances and compare the results of the new methods directly with the results reported in literature on these same instances.

In what follows we detail our implementation of a benchmarking simulation framework which we call FabricatioRL. We achieve this in two steps. First we use the standard put forward in Chapter 2 to derive requirements for an RL benchmarking simulation framework for production scheduling problems in terms of scheduling setup, RL control interface and reproducibility (Section 3.1). Then we show how these requirements can be satisfied by describing the design and implementation of FabricatioRL, with a particular emphasis on its inputs, API, and core logic (Section 3.2). Before conclusing this section with an outlook on future work (Section 3.4), we describe a visualization app associated with the simulation together with two simple framework usage examples (Section 3.3).

3.1 Requirements

The pivotal role requirements play within the context of software engineering has been a generally accepted fact for decades (Bell et al., 1976; Karlsson, 1996; Hussain et al., 2016). In the present case, accounting for the particularities stemming from the overlap between the production scheduling and RL fields is crucial to ensure the overall effectiveness of our work. By defining the simulation's goals, scope, and constraints, we ensure a tight domain fit, thus increasing the acceptance potential of our framework while reducing the implementation overhead. This requirement analysis and implementation task is made particularly challenging by the high degree of variance with respect to scheduling setups and RL designs demonstrated in the previous chapter.

Hereafter, we distinguish between functional (f) and non-functional (nf) requirements. In terms of the functional requirement definition, the literature reveals a broad consensus (Glinz, 2007): They define the functions that a system must be perform, i.e. the system scope and goals (IEEE Computer Society. Software Engineering Technical Committee, 1983; Glinz, 2007; Robertson et al., 2012). However, this consensus does not apply with respect to non-functional requirements. We choose to see non-functional requirements as "attribute[s] of or constraint[s] on [the] system" (Glinz, 2007). Essentially, functional requirements specify *what* the system does, while non-functional requirements mandate *how* the functions should be performed.

We use the standard put forward in the Chapter 2 to derive requirements for our simulation framework. Following our standardization framework's structure yields three requirement categories. First, we discuss requirements associated with scheduling setups in Section 3.1.1. Secondly, in Section 3.1.2, we focus on requirements ensuring the smooth interaction of our framework with RL algorithms. Last but not least, we detail the requirements stemming from our experiment validation standard in Section 3.1.3.

3.1.1 Scheduling Setup

The following requirements arise from the discussion of scheduling setups in Section 2.2. Functionally, FabricatioRL should target the more general production scheduling problems in terms of base setups and additional constraints, and should allow for the monitoring of varied scheduling targets. The non-functional requirement associated with setups is modularity.

Generality (f): The setups considered in RL literature show just how varied production is. However, different α and β are not independent of each other. Rather, there is a subsumption relationship between them, meaning that setups with more generic constraints subsume setups with stricter constraints. This is most obvious for the α-parameters. While less evident, the subsumption relationship also exists for some of the β-values. We require the benchmarking framework to cover a large number of typical scheduling setups, or, stated differently, to be as general as possible.

Because of the subsumption relationship defined by α (see Figure 2.1), implementing the flexible partially open shop setup ($FPOc$) would lead to the coverage of 91% of the setups studied in the investigated RL literature. Having implemented this setup, the simulation could then be instantiated to any of the setup's sub-classes. Additionally, machine and job related speeds (Rm) should, ideally, be implemented as well. In doing so a 100% coverage of the literature setup would be reached.

In terms of β, the set consisting of the following eight parameters yields a 77% literature coverage at a manageable implementation cost:

1. variable number of operations ($vnops$),
2. input buffers ($block_{\text{in}}$),
3. stochastic processing times (p^s_{ji}),
4. transport times ($tr(\infty)$),
5. operation capabilities M^o_i,
6. resource- and sequence-dependent setup times (s_{jik}),
7. stochastic release times (r^s_j), and
8. machine breakdowns ($brkdwn^s$).

Because of the their more sparse subsumption relationship (see Figure 2.2), β-parameters induce much more implementation work. To obtain the full coverage of the investigated RL scheduling work, r^s_j would need to be replaced by the stochastic demand (dmd^s_j), and ten more parameters would need to be implemented, namely:

9. permutations ($prmu$),
10. job precedence ($prec$),
11. output buffers ($block_{\text{out}}$),

12. job batches (j_{batch}),
13. flexible resources ($fres$),
14. stochastic capacity transports ($tr(k,r)^s$),
15. transport setup times (s_{ji}^{tr}),
16. forced parallel operations ($fpops$),
17. dynamic batches ($dbatch$), and
18. partial no wait ($pnwt$).

This would indeed be a daunting task.

We deem the following setup to be sufficiently general, since, it subsumes most of the setups described by Pinedo (2012) and the RL literature: ($FPOc|vnops, block_{\text{in}}, p_{ji}^s, tr(\infty)$, M_i^o, s_{jik}, r_j^s, $brkdwn^s$). This means that the job operation precedence is described by a DAG and operations can be executed on one or more available machines. The number of operations in every graph can differ, and particular operations can occur more than once within the same job. The processing speed is machine dependent and is used to scale the duration associated with every operation. Every operation has an associated tool set, and the tool switching times are sequence-dependent. Transport times are modeled explicitly but it is assumed that sufficient vehicles are available so as to immediately start every transport task. Buffers of a certain capacity (including 0 and ∞) are placed before each machine. machine breakdowns, job arrivals, and operation processing times are stochastic. For a better readability, we refer to this scheduling setup as Generalized Flexible Job Shop (GenFJS).

The selection of the parameters to exclude was driven by practicality. We do not implement the Rm setup since it is seldom used and requires the inflation of operation duration matrices along a third dimension, which implies memory overhead. In terms of β we mostly disregard one off parameters, e.g. $fpops$, $fres$, which we consider fringe cases. There is one exception to this rule, which concerns the explicit modeling of vehicles, i.e. the setups subsumed by $tr(k,r)^s$. There are 18 works considering some variant of these parameters, which amounts to 11% of all the encountered βs. While significant in terms of coverage, the overhead associated with implementing two additional decision types (see Figure 2.6) on top of the two we already consider, (see Figure 3.1) was deemed to high for the first version of the simulation framework. Note that the job routing decisions are induced by the presence $FPOc$ as an α-parameter, or the presence of M_i^o in the β-parameter-set or both.

γ-Traceability (f): γ defines either job- or resource-centric optimization goals. The most frequent job-centric goals are aggregates (e.g. maximum, average, minimum) of job completion times, flow times, throughput times, lateness, tardiness, earliness and job idle times. Resource-centric goals aggregate resource utilization, number of operations in buffers, buffered processing times, incurred setup times, machine failures and inventory levels. The simulation should track all of these intermediary variables at every time-step,

such that the desired target, e.g. minimizing the maximum completion time (makespan), can easily be measured by applying a corresponding aggregation function. To this we refer as γ-traceability.

Furthermore, the intermediary variables in Figure 2.4 should all be available, such that custom optimization targets can be easily implemented and experimented with.

Modularity (nf): To ensure that our framework can be expanded upon in the future, we require it to be modular. On the one hand, we do not implement all the setups that we encountered while assessing the state of the art. On the other hand, there may be many more production details that will be considered hereafter, as computing resources become more readily available and production itself becomes more diverse. For these reasons it is essential, that the simulation have a modular design, such that new constraints can be mapped to simulation logic with as little overhead as possible.

3.1.2 RL Modeling

Because of the many options for RL design, the first functional requirements related to RL modeling (see Section 2.3) revolve around configurability with respect to the action, state, and reward-spaces. Additionally, FabricatioRL must be compatible with available RL agent libraries. From a non-functional point of view, the simulation needs to be asymptotically efficient in terms of runtime.

Reactive Breakdown Coverage (f): In terms of MDP breakdown, FabricatioRL should mainly focus on purely-reactive breakdowns. Because of the significat differences between the 11 breakdowns introduced in Section 2.3.1, more than one simulation paradigm is required to accommodate them all. Since the six purely-reactive breakdowns make up 73% of the investigated literature and the reactive paradigm best fits the adaptivity argument used for RL scheduling, we use this breakdown category as a reference point in discussing how to accommodate the the others.

Simulations for purely-reactive breakdowns are orthogonal relative to the Iterative Gantt Improvement breakdown. On the one hand, purely-reactive breakdowns require an environment that accepts direct shop-floor decisions (operation indices etc.) based on a shop-floor state, when resources are freed. Within Iterative Gantt Improvement breakdown, on the other hand, RL agents operate directly on (partial or broken) scheduling solutions, i.e. the Gantt-chart representations. Given the orthogonal nature of the environments expected by the two breakdown classes, at least two distinct simulation paradigms are required to cover all the experiments in literature.

A well designed purely-reactive breakdown simulation could be used as a core component for the Iterative Search Refinement breakdown. A key component of a Iterative Search Refinement breakdown simulation is a scheduling solution checker. Consider for instance the approach by Cao et al. (2021). Recall that in the referenced work, the agent is presented with EA population features (state) and is expected parameterize the next iteration of the algorithm by means of an action. The solution checker can be used both for generating the raward signal between iterations and for calculating the fitness for individuals within the

population. However, a wrapper is required to extract population features and control the RL agent-action-guided population evolution.

Different from the other two previous breakdowns, a purely-reactive breakdown environment can be used to train agents following the Iterative Edge-Definition, Direct Planning, and Iterative Re-Planning breakdowns. The Iterative Edge-Definition case can be modeled as a purely-reactive breakdown. The addition of disjunctive edges to the graph representation of the schedule could be triggered when machines become available. The node in the graph corresponding to the machine requesting a decision is then marked for the agent, who is additionally presented with the partial graph corresponding to the current state. The agent is then limited to adding edges for that machine only. In the Direct Planning case, given that the agent produces the schedule in one pass based on an initial state, i.e. the list of jobs to be scheduled together with their properties, a purely-reactive breakdown environment simply needs to be capable of executing and evaluating the schedule produced by the agent. The same holds true for Iterative Search Refinement. Here the simulation can be used to inform the agent of the quality of the solutions present within the EA populations, for example. For Iterative Re-Planning compatibility, the purely-reactive scheduling simulation needs the capability to follow a predefined operation sequence plan and request a new decision on occurrence of stochastic events.

In keeping with the pragmatic approach of covering as much of the reviewed literature experiments with as little implementation overhead as possible, we require FabrikatioRL allow for the easy instantiation of all the purely-reactive MDP breakdowns accommodating Sequencing on Machines (i) and Job Destination Selection decisions (ii) (see Table 2.1 and Figure 2.6). This leads to the exclusion of the Transport-Centric Sequencing[1] and Interlaced Tooling and Sequencing breakdown. Despite it, the proposed requirement, which we call Reactive Breakdown Coverage for concision, still accommodates 68% of the breakdowns in literature.

The most important event to a reactive production scheduling MDP is that of a resource finishing its current task. For GenFJS, two questions arise on such an occurrence: "What operation should be processed next on the resource just freed?"(i), "To which downstream resource should the job just processed be sent?"(ii).

The (i + ii) sparation of scheduling decisions leads to the process flow depicted in Figure 3.1: Whenever the operation of a job is finished on a machine, an RL agent selects the next operation for this machine from the input buffer (i). Then, the agent selects the transport destination (ii). The possible destinations for the job just processed depend on the next eligible operations and the machines capable of processing them. The operation feasibility can be derived from the precedence graph depicted in the top left corner. Depending on the particular scheduling problem and its assumptions, not all decision types will be encountered. For standard job-shop scheduling problems, for instance, only decisions of type (i) will be required.

[1]Note that our definition of the generality requirement as a simulation implementing GenFJS is also not sufficient for instantiating Transport-Centric Sequencing breakdowns.

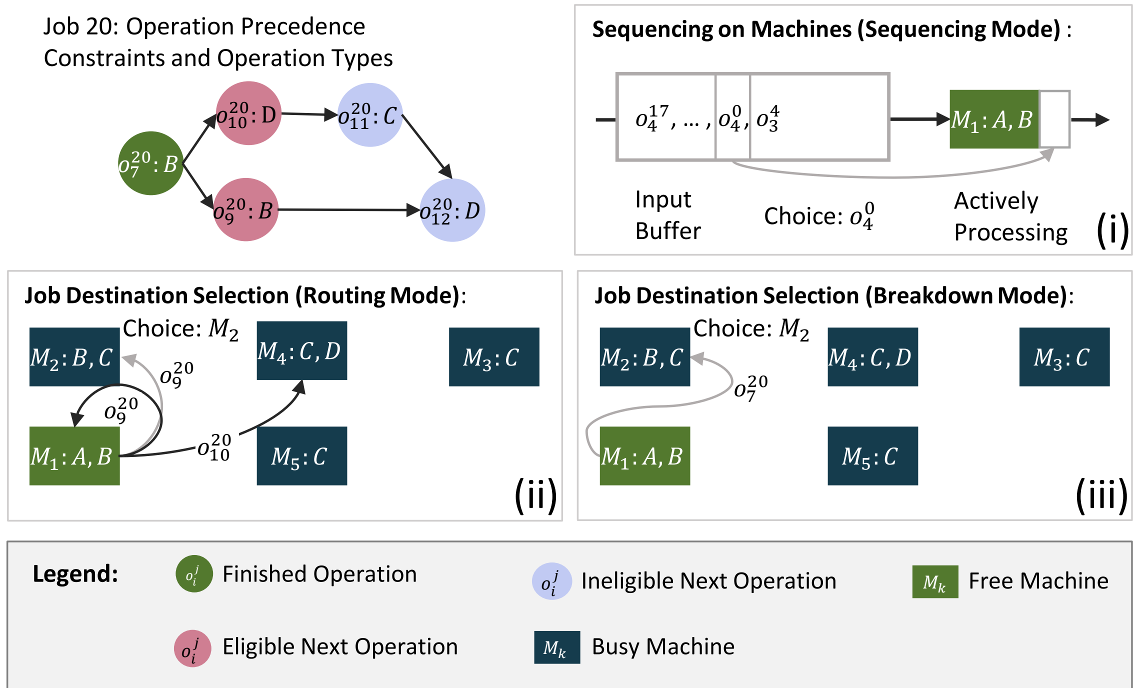


Figure 3.1: The two decision types in a production setup with operation precedence graphs (top left) and machine breakdowns. Operation and machine types are indicated with capital letters. In (i) and (ii), gray arrows model decisions made, and blue arrows represent alternatives. In the case of breakdowns (iii) all operations buffered at the failed machine need to be routed away; Should an operation execution be interrupted by the breakdown (o_7^{20} above) it too needs to be routed away. The three different decisions correspond to different simulation modes which will be detailed in 3.2.4.

Machine breakdowns within contexts with job routing flexibility also induce job decisions (iii). In this case, all the operations buffered at the failed machine need to be moved away to alternative processing stations. Should these alternatives not exist, then the operations remain blocked at the failed resource until its repair.

Each decision in Figure 3.1 is associated with a different simulation mode. Depending on the mode, decisions taken by RL agents will be interpreted and handled differently: In Sequencing Mode, agent actions will be interpreted as operation indices, while in the routing, and breakdown modes actions correspond to machine indices. The difference between the routing and breakdown modes is given by the position of the operation transported. The mode details will be elucidated in the subsequent implementation section (see Section 3.2.4).

Action-Space Configurability (f): There are two main approaches to encoding the actions described above. When faced with a decision, an agent can either select an action directly or indirectly. Indirect actions are chosen by selecting an optimizer from a fixed set, which then determines the next direct action. Direct action for (i) comes in the form of an operation index, i.e. the tuple (job number, operation number), from the set of all available operations in production. For (ii) and (iii) the action is given by a machine index. Additionally, the action can be deferred by outputting a wait signal. Examples of direct decisions can be found in the elaborations of Jiménez (2012), Qu et al. (2015), and Kuhnle et al. (2020),

while indirect decisions are covered by Aydin et al. (2000) and Luo (2020), for instance. Optimizers are often, but not always (e.g. Shahrabi et al., 2017), simple priority rules.

For the simulation framework, these alternative approaches lead to the action-space configurability requirement. The simulation should allow both direct and indirect actions. Additionally, it should be configurable whether the RL agent is responsible for all decision types, or just a subset, deferring action to fixed optimizers for the rest. The optimizer sets for indirect actions and RL complementary action should be easily customizable and extensible.

State-Space Configurability (f): The information needed for state-transition depends on the production setup considered: The state tracks the jobs currently in production together with the operations still in need of processing, their remaining processing time, position (machine index) and status (in transport, waiting for processing, processing). Additional fixed information such as transport times, tool switching times, precedence constraints, machine capabilities, buffer capacities, operation types and tool sets should be available, such that the setup constraints can be enforced by the simulation and learned by the agent.

In RL, a distinction is made between agent-state and environment-state. The latter contains all the information required for the environment to implement its logic, while the former is an agent view of the latter. For production scheduling, the agent-state often only contains a subset of the information available in the environment-state. This is because the environment-state-space may be too large for the agent algorithm to handle. Furthermore, simulated stochasticity is transparent to the environment, but should not be transparent to the agent. Lastly, it may be useful to have agents make good decisions based solely on environment-state features that are independent of problem size. This would improve the agent's transferablility and generalization qualities.

The exact information comprising the agent-state needs to be established through experimentation. On the one hand, raw environment-state information as listed above, can be used. On the other hand, environment-state information can be condensed into features. Features fall into three categories, namely job features, resource features or target features. The first category aggregates job information, e.g. remaining job processing time or remaining job operations. The second aggregates machine or vehicle related information, e.g. remaining processing time in machine input buffers or number of operations queued for processing or transport. Information in these categories can be stored per job/machine or aggregated further into single scalars using centrality (e.g. mean, median) and variance measures (e.g. standard deviation, gini). The last feature category contains optimization goal related variables such as estimated total tardiness (Wang et al., 2005; Luo, 2020) or average machine utilization (Thomas et al., 2018; Luo, 2020).

The simulation framework should allow the selection of the agent-state components and allow extensions thereof. These can be either raw state information such as the current operation duration matrix or operation position, as well as different features including goal oriented metrics. Moreover, it should be made possible to accommodate user defined

state features.

Reward Configurability (f): There is no generally accepted scheme for reward design, which means that appropriate signals have to be found through experimentation. Since RL agents try to maximize future reward, it stands to reason that, for production scheduling, the reward is often a function of the optimization goal or a goal-related intermediary variable, (e.g. Qu et al., 2015; Wang, 2020; Luo, 2020; Kuhnle et al., 2020). Important choices in reward design include the time points at which the reward is returned (at every step, every k steps for some k or at the end of the game), whether the reward is continuous or discrete, strictly positive, strictly negative or both, bounded or unbounded (Sutton et al., 2018).

Extended Gym Compatibility (f): In terms of the agent interface, the simulation should respect the OpenAI Gym standard, such that external agent libraries such as keras-rl can be used in conjunction with FabricatioRL. This allows the application of different pre-implemented RL agents, whether they be policy-based, value-based or actor-critic systems, to be trained and tested within the environment in a convenient fashion.

Additionally, the simulation should address two supplementary RL techniques not currently covered by the gym standard, namely illegal action masking and offering an environment clone for model-based RL approaches such as AlphaZero (Silver et al., 2017b). To construct action masks, the environment has to provide the agent with a list of legal actions at every step. Environment clones can be used by agents to directly see the effects of one's actions a few steps ahead.

Runtime Efficiency (nf): A simulation requirement that does not follow directly from the details laid down until now is that of simulation performance in terms of runtime. RL is sample-inefficient (Haarnoja et al., 2018), which implies that many simulation runs will be required for the agent(s) to converge. Thus, to increase the ease of experimentation, we should strive to decrease the runtime as much as possible.

3.1.3 Validation

For the purpose of allowing for easy scheduling approach validation (see Section 2.4), FabricatioRL needs to implement one non-functional and two functional requirements. From a non-functional point of view, our simulation framework should allow for the reproduction of stochastic experiments. Functionally, FabricatioRL should ensure the separation of scheduling inputs from the scheduling process itself, and be compatible with not only RL, but also planning methods.

Reproducible Stochasticity (nf): Recall that, while the reproducibility requirement is not bound to the domain of stochastic scheduling, in the RL scheduling literature, this essential requirement was often neglected (see Figure 2.18).

For (online) stochastic problems, the general literature approach is to sample stochastic events *on demand* during the simulation. For the purpose of ensuring the experiment reproducibility, solely reporting the distributions used to simulate stochastic influences, as

is being done in literature, is not sufficient. Consider, for instance, a scheduling problem with 10 jobs and 5 machines. Let the job operation sequence be defined by a random permutation of numbers 1 to 5. This leads to $5!^{10}$ possible scheduling instances. The enormous sample space coupled with a relatively small number of experiments leads to an extremely low probability of sampling similar instances a second time. Because of the probable difference in sample composition, a reliably comparing the results of new approaches to those reported in literature is virtually impossible. To prevent this problem, a variance reduction technique (see Yang et al., 1991) should be implemented.

To control stochasticity, the common random numbers (Glasserman et al., 1992) approach can be used instead. This means that (a) the sequence of random numbers required during the simulation is a function of a single independent variable and (b) the meaning of drawn random numbers is the same between simulations. We refer to the requirement of implementing common random numbers as "reproducible stochasticity".

Input Separation (f): By fulfilling the reproducible stochasticity requirement and publishing the simulation framework code, experiment reproducibility could be bound to the use of FabricatioRL. Consider the following sampling scheme for a $(Jm \mid r_j^s)$ setup, for example: At the beginning of the simulation, we sample job release times $r_1, \ldots, r_n$. Then, whenever the simulation time reaches r_k, the operation type and operation duration sequences defining a new job are sampled, and the job is added to the system. While the use of RNG seeding along with the publication of experiment-associated seeds would enable researchers to reproduce the associated results when using our framework, running identical experiments using an alternative simulation framework would not be possible.

To curtail this problem, we require that the simulation inputs, including the instantiation of stochastic events, be clearly separated from the rest of the run. Within the frame of the previous example, this would mean that both the release time and the jobs would need to be sampled during the simulation initialization phase. The inputs can then be saved and used to parameterize arbitrary alternative simulations. We refer to this requirement as input separation. As an added bonus, the input separation requirement also ensures that RNG-generated nonce sequences would semantically map to the same variables, e.g. the nonce at position k in the sequence would always map to the duration of operation i from job j, independent of the control algorithm interacting with the simulation.

Backwards Compatibility (f): Seen as many experiments detailed in Chapter 2 relied on preexisting benchmark scheduling problem instances, we require that our framework can be instantiated to such instances as well. We dub this requirement "backwards compatibility" and specify that the production scheduling instances made available by Beasley (1990) (OR library) and Mastrolli (1998) be available through FabricatioRL as well.

Planning Compatibility (f): Lastly, the simulation should allow following an externally computed production plan, where the operation start times are listed for every resource. Such an execution plan is often the output of more traditional OR exact re-planning

approaches, e.g. MILP, CP. While such an a priori plan may be affected by stochastic events, it may still yield better results than RL in many situations. The exact situations where RL outperforms such OR approaches have yet to be established. Our simulation framework should enable researchers to cover this gap, which leads to the planning compatibility requirement.

3.2 Implementation

FabricatioRL is written purely in Python and contains no significant external dependencies outside of numpy and Gym. In particular, we use no discrete event simulation libraries such as SimPy (Müller et al., 2020) to maintain full control over all the simulation framework's structures, which makes the task of guaranteeing reproducibility more manageable. We chose Python since this integrates seamlessly with both Gym and the most widespread RL agent libraries. Additionally, one of the best performing CP frameworks, namely ORTools (Perron et al., 2022), can also be easily embedded within Python code.

Note that it is not the purpose of this elaboration to fully describe the implementation. Rather we strive to describe just enough to make it easier for the reader to dive into our code. We focus somewhat more on the simulation functionality contained by the FabricatioRL class located in the interface module, which contains the simulation entrypoint. This is because a good understanding of the API functions will enable researcher to make use of FabrikatioRL without understanding all its innerworkings. However, if the reader's purpose is the extension of the simulation, a deep dive into the code cannot be circumvented.

We start the implementation discussion with a very brief presentation of FabricatioRL's architecture in Section 3.2.1. This section offers a high-level overview of the simulation framework's main structures, anchoring the implementation details that follow. We encourage the reader to revisit this overview whenever a re-contextualization of particular simulation framework elements is required. In Section 3.2.2 we discuss the FabricatioRL's initialization process and its many possible parametrizations (inputs) along with associated sampling functionality leading to the generation of varied production scheduling setups. Section 3.2.3 details FabricatioRL's API with a particular emphasis on RL design configuration. Last but not least, in Section 3.2.4 we turn to the core module, focusing our elaboration on the simulation state structure and main simulation loop. At the end of each of the last three implementation sections note how the particular implementation aspects contribute to the satisfaction of the previous section's requirements.

3.2.1 Architecture

Figure 3.2 shows the main components of the layered architecture of FabricatioRL following the Gym API. The Gym API consists of three core methods namely init, step and reset. init is used for parameterizing and instantiating the simulation. step takes an action as an argument and returns a state, the reward for the particular state, a flag signaling whether the simulation has ended and a dictionary with debugging information. reset is called to

move the simulation back to its starting state. The render is an optional method defined by Gym to create a visual representation of a state. This last method is not implemented by FabricatioRL though we do list it here for completion.

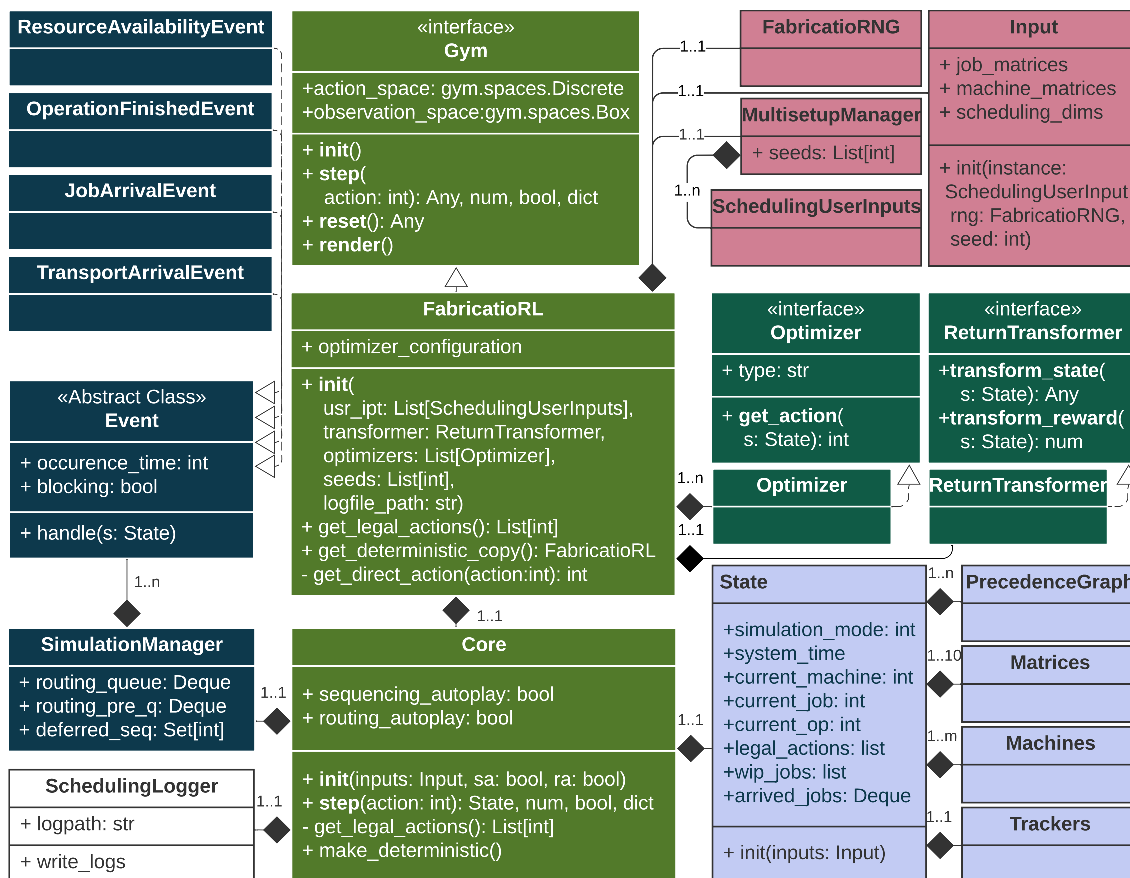


Figure 3.2: Class diagram of the environment architecture.

FabricatioRL separates the main simulation functionality located in Core module from the FabricatioRL wrapper, which implements the Gym interface. The layer components are highlighted in light green. The red classes pertain to the definition and storage of simulation inputs. The dark green classes deal with customizing FabricatioRL's interface in terms of actions, state representation and rewards. The dark and light blue highlighted classes represent the two main data structures of the environment, namely SimulationManager and State together with their respective components.

3.2.2 Initialization and Inputs

Initialization: FabricatioRL's initialization process is depicted in Figure 3.3. The sequence diagram visualizes the communication between the FabricatioRL, FabricatioRNG, MultisetupManager Input, Core, SimulationManager and State classes, i.e. all the main simulation components. Note that all classes depicted in Figure 3.2 are, in fact, involved in the process. Nevertheless, we excluded the details of the SimulationManager, State, and SchedulingLogger initialization for concision. First FabricatioRL.init is called with five parameters, namely user_inputs, transformer, optimizers, seeds and logger_path.

The parameters passed to the Fabricatio.init (0) call are in part defined by the simulation

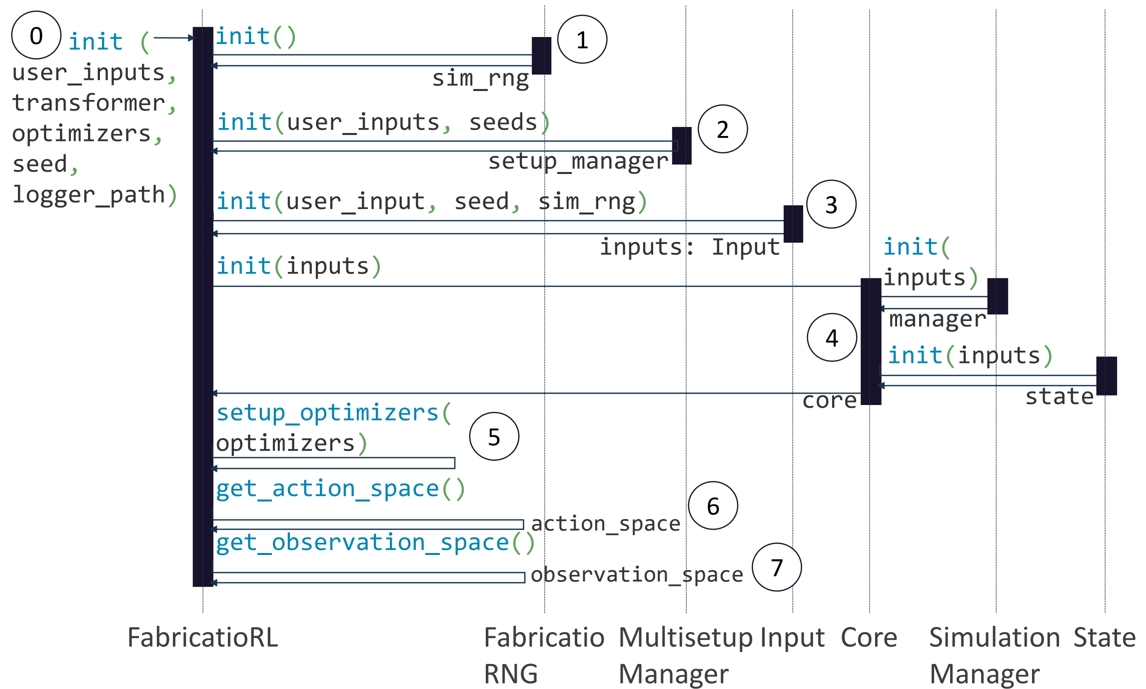


Figure 3.3: Sequence diagram of FabricatioRL's initialization process.

itself, within the interface_templates module. The user_inputs parameter is a list of SchedulingUserInputs types. This class defines and documents the parameters that can be used to define a GenFJS instance or an instance of any of the other subsumed problems. The transformer and optimizers objects, which can be used to modify FabricatioRL's interface, must implement the API defined by ReturnTransformer and Optimizer respectively. The seeds parameter is expected to be a list of integers to be used as RNG seeds. Finally, the logger_path is a string pointing to a folder where logfiles are to be written.

There are seven main steps involved in the simulation initialization. The first three steps pertain to defining the simulation instance. The fourth step's purpose is the initialization of the core logic modules, while the final three steps pertain to FabricatioRL's API configuration.

In step one, the simulation's own RNG is initialized (1). Initializing FabricatioRNG returns a numpy RNG instance distinct from the one used internally by the library. In separating between our own RNG and numpy's, we ensure that seeding the numpy RNG outside our simulation does not affect the sampling sequence within FabricatioRL. In step two, the list of user inputs are passed to the MultiSetup initialization function (2). FabricatioRL can cyclically run on multiple distinct instances, which can be useful for training and/or evaluating models with transferability in mind. The MultiSetupManager's job is to create the sequence of unique instances though which FabricatioRL cycles as the Cartesian product of the scheduling input and seed sets. In step three, the first pair of user_input-seed objects together with the sim_RNG are used to initialize an Input object (3). Depending on how the user defined the user_input object, random sampling may be required to generate the full scheduling instance, which is why the RNG is used. The Input class encapsulates all scheduling instance information.

Step four involves using the freshly assigned inputs attribute to initialize the core object which is stored within an attribute of the same name in the FabricatioRL object (4). The core initialization itself consists of instantiating and storing the central core data structures, to wit the SimulationManager and State objects.

Having instantiated the core, the initialization process moves to the final steps. In step five, the action-space configuration, is inferred during the setup_optimizers from the number and type of optimizers passed to FabricatioRL.init (5). The action-space configuration defines the simulation's action interpretation in terms of direct or indirect actions (see Section 2.3.2) for the different decision types, i.e. regular job routing, sequencing, and failure job routing. The get_action_space call in the sixth step then initializes the action_space property defined by the gym interface to the appropriate dimension (6). Similarly the observation space dimension is inferred using the ReturnTransformer parameter in the final step (7). If no such object is passed, the simulation simply sets the observation_space property to None. The next section offers more details on the possible API parametrizations induced by the Optimizer and ReturnTransformer objects

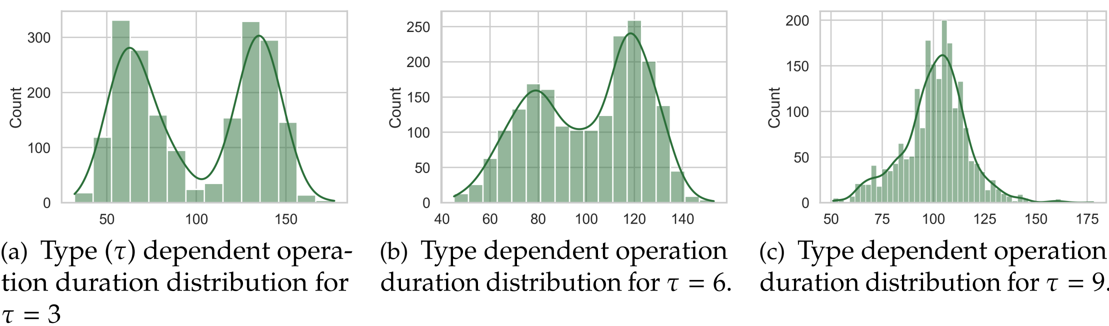
Inputs: The following scheduling instance information can be specified by instantiating SchedulingUserInpusts appropriately. Let n, m, o, l, t be the number of jobs, number of machines, maximum number of operations per job, number of system tool sets and number of operation types respectively. The inputs for GenFJS are given by the operation precedence graphs $O^P \in \{0,1\}^{n\times o\times o}$, the operation type matrix $O^{Ty} \in \{1,\dots,ty\}^{n\times o}$, the operation duration matrix $O^D \in \mathbb{N}_+^{n\times o}$, the operation tool set $O^{Tl} \in \{1,\dots,tl\}^{n\times o}$, the machine speed vector $M^S \in \mathbb{R}^m$, the machine distance matrix $M^{Tr} \in \mathbb{N}_+^{m\times m}$, the tool switching time matrix $M^{Tl} \in \mathbb{N}_+^{l\times l}$, the machine input buffer size vector $M^{Bf} \in \mathbb{N}_+^m$, machine capability matrix $M^{Ty} \in \{0,1\}^{m\times t}$, the maximum number of failures per machine f, the distributions for the time between failure ϕ, the operation duration noise δ and the job inter-arrival time ι. The operation duration noise $O^{D'} \in (0,2)^{n\times o}$, job release times $R \in \mathbb{R}_+^n$ and machine breakdown times $B \in \mathbb{R}_+^{m\times f}$ get sampled in accordance with δ, ϕ and ι respectively. A due date vector can be passed to the simulation $T \in \mathbb{N}^n$ defining job due dates. B and R are used to create the stochastic events to be added to the event queue in Core during its instantiation. To run the simulation on instances of problems subsumed by GenFJS, the unnecessary inputs can be simply left out.

For dynamic problems, FabricatioRL uses a WIP, which means that the WIP size $w < n$ must be additionally specified. Furthermore, the number of initially visible jobs $n' < n$ must be provided in such cases.

FabricatioRL offers three ways of defining the matrix inputs. First, the simulation user can choose to pass all the listed matrices and vectors directly to the FabricatioRL.init function. Secondly, the different input dimensions, e.g. the number of jobs and the number of machines in a Jm setup, together with or without a corresponding sampling function can be provided. Sampling functions are expected to take a shape as a parameter and return an numpy array of corresponding size filled with the desired entries. When dimensions are provided without the sampling functions, FabricatioRL uses its extensive default sampling functionality to fill the gaps. Finally, a path to a standardized scheduling instance (e.g.

Beasley, 1990) can be provided. In such all the job and resource information contained in the respective instance, is extracted from the file. Dynamic scheduling setups can be created based on instance files as well. The job information contained in the file will be uses as a sampling pool in such cases. The desired n total jobs will then be sampled uniformly at random with replacement from the file based pool. Currently only Jm and FJc instances are supported for the file-based instantiations.

Sampling Schemes: The random sampling functionality for operation types O^{Ty} and durations O^D is connected. Recall that for a machine to execute an operation, the machine and operation type must match. Operation type sampling depends on the desired setup. FabricatioRL generates Jm and Fm style job types using n different random permutations of numbers 1 to m for the former case and a single random permutation for the latter. Completely random types can also be generated, e.g. for $vnops$ setups, by sampling each operation type from $\{1, \ldots, ty\}$. Once the types have been defined (either through sampling or using the user input directly), duration distributions are created separately for each type by summing up two lognormal distributions, namely Lognormal($50 + 50 \cdot \tau/ty, 0.2$) and Lognormal($150 - 50 \cdot \tau/ty, 0.08$), where τ is the operation type. This generates bi-modal distributions with both modes moving towards each other as τ increases until a uni-modal distribution is reached (Figure 3.4). This synthetic data generation scheme, though not rooted on empiric cases, models the intuitive dependency between operation type and duration. The distribution modes are loosely based on the OR library benchmark operation duration averages.



(a) Type (τ) dependent operation duration distribution for $\tau = 3$

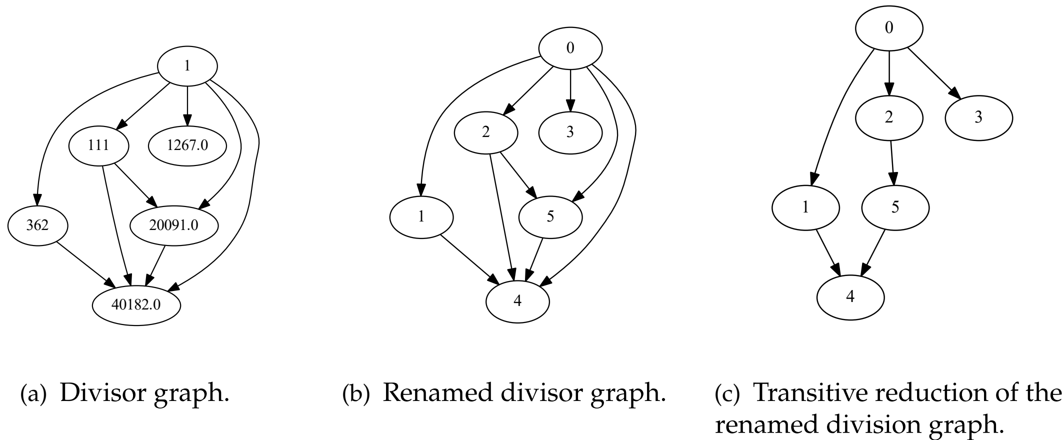
(b) Type dependent operation duration distribution for $\tau = 6$.

(c) Type dependent operation duration distribution for $\tau = 9$.

Figure 3.4: Examples of the type τ dependent operation duration distributions $Z = X + Y$ where $X \sim$ Lognormal($50 + 50 \cdot \tau/ty, 0.2$) and $Y \sim$ Lognormal($150 - 50 \cdot \tau/ty, 0.08$) for $ty = 10$ total operation types.

The random operation precedence generation process is the most sophisticated among the sampling schemes. Figure 3.5 visualizes the generation process for a single job with $o = 5$ operations. First a random integer between 1 and $1e^6$ is sampled uniformly at random, its divisor set is computed and the graph of the division relation is created (Table 3.5a). If the numbers in the divisor set has a cardinality larger than $o + 1$, divisors are removed from it until only $o + 1$ integers remain. Conversely, if there are not enough divisors in the set, a second integer is sampled, its divisor set computed and added to the prior set. While adjusting the divisor set cardinality, we ensure that one always stays contained in the set. In a second step, the nodes in the division relation graph are renamed to using sequential integers starting at one and ending with o (Table 3.5b). One is renamed to 0.

Finally, we compute the transitive reduction (Aho et al., 1972) of the renamed division graph to eliminate any redundant edges (Table 3.5c). The resulting graph represents the desired random precedence. The 0 node represents a dummy root node with its edges pointing to the job operations that can be processed first.



(a) Divisor graph. (b) Renamed divisor graph. (c) Transitive reduction of the renamed division graph.

Figure 3.5: Stages of DAG generation.

If indicated by the user, transport times M^{Tr} and tooling times M^{Tl} are sampled based on the values in O^D such that transports constitute an overhead comparable to operation processing. The minimum, maximum, mean and standard deviation of operation durations are first extracted from O^D. These values are then used to parameterize a truncated normal distribution from which a symmetrical distance/tooling time matrix is sampled.

Release dates R, breakdown and repair times B and due dates T have no associated sampling scheme. The first two need to be defined by the user. For the due date computation, a float defining due date tightness is expected as a parameter. This number is then multiplied with the sum of operation durations in every job resulting in the job due dates. Note that for release dates, the scheme described in 6.1.2 will be implemented within FabricatioRL in the future.

The sampling schemes associated with the remaining inputs M^{ty}, M^S and $O^{D'}$ are straightforward. Machine capabilities are sampled in two steps. For each machine, the number of capabilities m_{ty} is sampled uniformly at random from $\{1, \dots, ty\}$. Then m_{ty} types are sampled uniformly at random without replacement from the same type set. Machine speeds are sampled uniformly at random from the interval $[0.5, 1.5]$. There is no sampling functionality associated with the buffer length. Operation duration noise matrix $O^{D'}$ is sampled using a truncated normal distribution with a mean and standard deviation of 1, lower bound of 0.1 and upper bound of 2. This is quite extreme, since the noise is multiplicative and the described sampling scheme yields an almost uniform distribution of perturbation on the interval $[0.1, 2]$. Alternatively, a float z from the interval $(0, 1)$ can be used to parameterize the noise. In such a case the noise matrix entries are sampled uniformly at random from the interval $[1 - z, 1 + z]$

Requirements: *Reproducible stochasticity*, is achieved by seeding the dedicated Fabri-

catioRNG object. All sampling happens after seeding but before the beginning of the simulation execution. In doing so, running the simulation on a particular input set with the corresponding seed would always yield the same stochastic events irrespective of the scheduling method employed. Thus, a dataset of input seed pairs could be used for the validation step, analogous to the benchmarks of supervised learning.

After the initialization of the Input object, the full input set (O^P, O^{Ty}, O^D, O^{Tl}, M^S, M^{Tr}, M^{Tl}, M^{Bf}, M^{Ty}, $O^{D'}$, B, R, T) including stochastic variable instantiations is fixed. During the Core initialization, the Input information is added to the State object. The state information is then used to determine the next processing steps and their duration. For example, the processing duration d of operation i of job j on the machine k having a current tool set of p can be calculated as $d = M^{Tl}_{pO^T_{ji}} + O^D_{ji} \cdot M^S_k \cdot O^{D'}_{ji}$. While the State version of the input information is modified during the simulation to reflect the production process progress, e.g. by zeroing out job matrix positions corresponding to finished operations, the Input information remains untouched. Logging these inputs leads to the fulfillment of the *input separation* requirement.

FabricatioRL contains all the Jm and FJc instances collected by Beasley (1990) and Mastrolli (1998), together with the required conversion functionality (see Section 6.1.1 for more details on the instance format). Loading a benchmark instance only requires specifying its path in the corresponding input parameter. As such, our simulation framework in *backwards compatible*.

3.2.3 Interface Module

Action-Spaces: Given the direct and indirect action designs found in literature together with different types of decisions possible, the multitude of state representations and posible rewards (Section 2.3), there is a clear need for API configurability in terms of action, state and reward.

To configure the action-space, different types and numbers of Optimizer objects are meant to be used. RL agents could, for instance, focus solely on sequencing decisions (Figure 3.1 i), deferring the job destination selection (Figure 3.1 ii and iii) to Optimizers objects passed to the FabricatioRL constructor. Such is the approach taken by Kuhnle et al. (2020), where agents solely take job routing decisions. All sequencing decisions are being done by a hard-wired optimizer, namely the FIFO heuristic.

The architecture presented here is more flexible, with optimizers being any object implementing the Optimizer interface from the interface_templates module. The interface is very simple, consisting of a type string, which can be either sequencing or routing, and the get_action method. Said method is offered a read-only State object and is expected to return a direct action adequate for the decision type, i.e. a machine index or an operation index. Note at this point that the operation index is an integer and not a tuple, as one might expect. The integer does unravel to a tuple, however.

Depending on the number and type of the Optimizer objects passed to the FabricatioRL, during initialization, and the types of decisions inherent to the defined setup, one of 11

optimizer configurations is possible, each leading to a different action-space. The configuration is stored in the optimizer_configuration field of the entry point class. Figure 3.1 gives an overview of the optimizer configurations together with their numbers and meaning. In the absence of any optimizer, the the action-space size for job routing decisions is given by the number of machines in the system, m. For sequencing actions, the direct action-space is given by the number of operations within the WIP, $w \cdot o$. If both decision types are present in the system, the action-space size would correspond to the sum of job routing and sequencing actions, i.e. $m + W \cdot o$. Since all production scheduling action-spaces are discrete, the action-space would be represented as a vector of size $w \cdot o + m$. Note that FabricatioRL expects the first action-space vector positions to correspond to sequencing actions and the last to correspond to job routing actions. The situation just described corresponds to the optimizer configuration 0, if the scheduling setup contains job routing flexibility and 1 if the scheduling inputs define a sequencing only setup.

Table 3.1: The 11 possible optimizer configurations influencing the action-space shape.

Number of Sequencing Optimizers \ **Number of Job Routing Optimizers**	0	1	> 1
0	optimizer-_configuration == 0/1 Sequencing: Direct Job Routing: Direct	optimizer-_configuration == 2 Sequencing: Direct Job Routing: Fixed Indirect	optimizer-_configuration == 3 Sequencing: Direct Job Routing: Indirect
1	optimizer-_configuration == 4/5 Sequencing: Fixed Indirect Job Routing: Direct	optimizer-_configuration == 6 Sequencing: Fixed Indirect Job Routing: Fixed Indirect	optimizer-_configuration == 7 Sequencing: Fixed Indirect Job Routing: Indirect
> 1	optimizer-_configuration == 8/9 Sequencing: Indirect Job Routing: Direct	optimizer-_configuration == 10 Sequencing: Indirect Job Routing: Fixed Indirect	optimizer-_configuration == 11 Sequencing: Indirect Job Routing: Indirect

Adding optimizers modifies the interpretation of the action-space vector. If a single optimizer of a particular type is added, the corresponding section of the action-space is eliminated. For instance, if a single sequencing optimmizer was passed to the simulation, but no job routing optimizers were defined (configuration 4), FabricatioRL will always expect direct job routing actions. Adding a single job routing optimizer and no sequencing optimizer yields to complementary behavior. In these cases the single optimizer is used by FabricatioRL directly when decisions of that particular type are encountered. The agent

will only be asked to pick the complementary direct action type.

When multiple optimizers of the same type are added, the action-space vector segment corresponding to that type will have a length equal to the number of corresponding optimizers. Take, for instance, configuration 8, and assume n_{so} optimizers of type sequencing were passed to the simulation. Here we have a setup with job routing flexibility, where multiple sequencing optimizers have been defined, but no job routing optimizer is present. Whenever job routing decisions are encountered, the agent will have to select one of the machines directly, which will correspond to indices $n_{so}, \ldots, n_{so} + m - 1$ of the action-space vector. m is the number of machines in the system. When sequencing actions are encountered, the agent will be expected to output the index of one of the sequencing optimizers, i.e. positions $0, \ldots, n_{so} - 1$ of the action-space. All other optimizer combinations impact the action-space analogous to the listed examples.

State and Reward Spaces: The ReturnTransformer interface defines an object which allows the customization of both the agent-state-space and the reward-space. The class located in the interface_templates module defines two functions namely transform_state and transform_reward for this purpose. Both functions are passed the complete environment-state as a parameter. The transform_state function is expected to select and format state information presented to the agent in a way that can be consumed by the latter. Currently only Box action-spaces are supported by FabricatioRL, meaning that the state must be a (multi-)dimensional numpy array. The transform_reward function works analogously: Given the entirety of the state information at a particular time, within transform_reward a numeric reward signal is to be constructed and returned. Note that it is possible for the class implementing the ReturnTransformer interface to define instance attributes allowing to save historic information that could be useful for constructing both rewards and state representation. Implementing either of the functions will require the user to familiarize himself with FabricatioRL's (nested) state structure.

Step: Figure 3.6 depicts the discussed structures working together during calls to the step function. In a first step, FabricatioRL decides how the action should be interpreted depending on the current optimizer configuration (1). If the agent action is to be interpreted as direct, e.g. opt_configuratio is 3 and the simulation_mode property of the state is 1, indicating that a job routing decision is required, then the control flow jumps directly to the second step (2.1). Otherwise, if the action is indirect, FabricatioRL executes get_action on the optimizer selected by the agent, passing the current state as a parameter (3). The get_action call results in the generation of an appropriate direct_action. In a second step, FabricatioRL's Core attribute executes its step method using the direct_action as a parameter (2.1)/(2.2), thereby moving the simulation into a new state, which is returned to the caller together with a flag that indicates whether the simulation has finished or not. Thirdly, and lastly, the returned State object is passed to the ReturnTransformer attribute which generates a state representation and reward signal as defined by the user within the implemented transform_state (4) and transform_reward (5) functions.

Reset: From the interface module's point of view, where the entry point class is located, the reset method can be seen as a combination of the init and step. Similarly to init, the

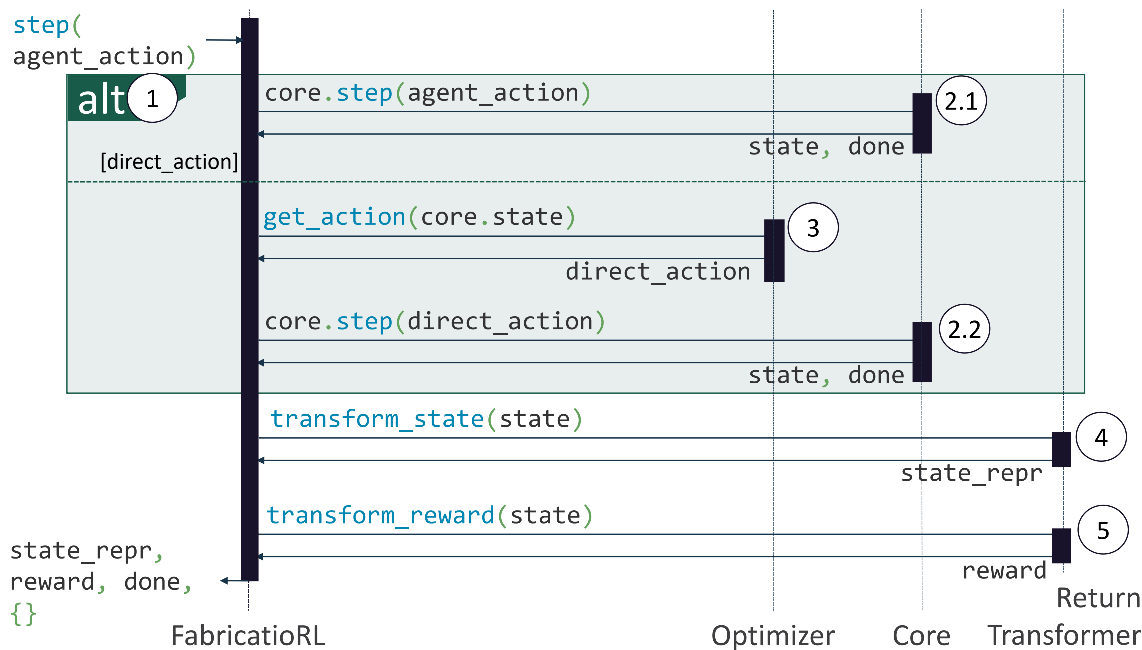


Figure 3.6: Sequence diagram of FabricatioRL's step function from the interface point of view.

SetupManager is used to get the next pair of user inputs and seed. These are used to re-initialize an Input object which in turn is used to re-initialize the simulation core. Similarly to step, the ReturnTransformer is used to transform the post re-initialization core.state into the representation defined by the user. This representation is then returned.

Legal Actions: To allow for the implementation of masking, the FabricatioRL class exposes the get_legal_actions method, which returns the action-space indices corresponding to viable actions. Illegal actions would lead to the termination of the simulation. The legal_action property maintained by the Core object in the State is used to infer the current legal direct actions. Indirect actions matching the current simulation_mode are all legal.

Consider the optimizer configuration three (Figure 3.1). The action-space in this case is a vector of size $n \cdot o + n_{ro}$ where the first $n \cdot o$ positions correspond to direct sequencing actions and the next n_{ro} positions represent indirect job routing actions. The variables n, o and n_{ro} represent the number of jobs in WIP the maximum number of operations and the number of job routing optimizers respectively. When the simulation is in job routing mode ($\text{simulation_mode} == 1$ or $\text{simulation_mode} == 2$), the returned legal actions correspond to the complete list of job routing optimizer indices $n \cdot o, \ldots, n \cdot o + n_{ro} - 1$. When the simulation is in sequencing mode ($\text{simulation_mode} == 0$), the legal actions will correspond to the those operation indices $i \in \{0, \ldots, n \cdot o\}$ which are currently buffered at the current_machine property of the state.

Deterministic Copy: The get_deterministic_copy function returns a variant of the simulation, where all stochastic events are removed from the state structure. More accurately, any jobs j with release times r_j larger than the state's system_time are removed from the simulation, the operation perturbation matrix $O^{D'}$ is replaced by a matrix with

all entries equal to one, and all the machine breakdown events are eliminated from the simulation.

This functionality is required by model based RL approaches such as AZ. In general, it is perfectly legitimate for any scheduling algorithm, RL or otherwise, to use the simulation for the assessment future states, as long as the stochastic information is not included.

Requirements: The described mechanisms lead to the fulfillment of the *RL configurability* requirement. By means of the Optimizer system, the action-space can be configured allowing for both direct and indirect actions. Using the ReturnTransformer object, the user has full control over what the agent can see in terms of states and rewards. Since the transformation functions both take a State object as a parameter, all the raw information, including tracking variables, is transparent, whereby full *state and reward-space configurability* is enabled.

The *gym compatibility* is satisfied by design, since the FabricatioRL class inherits from gym.Env and implements all three gym methods. By adding our get_legal_actions and get_deterministic_copy methods we satisfy the *extended gym compatibility* requirement. Section 3.2.4 will shed more light into the inner-workings the API functions from the core_perspective.

3.2.4 Core Module

The Core class contains the simulation logic expressed through the task of managing the event queue and state objects. Correspondingly, the main attributes of the class are the state of type State and the event management object, EventManager. Additionally, the class defines two autoplay attributes determining whether trivial decisions are skipped or not. If sequencing_autoplay is set to true, in sequencing decision cases where there is a single operation buffered at the machine will be skipped. Similarly, if routing_autoplay is set to true and there is a single viable downstream resource available for the job that needs to be routed, the decision will be taken by the simulation automatically. Note that turning on the autoplay function leads to limiting the space of all possible schedules. Sometimes it may be advantageous to defer processing an operation in favor of waiting for another operation to enter the machine input buffer, for example.

State: The State contains the all the information required for the simulation logic implementation. State information can be further grouped into four categories distinguished by the information's primary purpose. These are: agent information, convenience structures, WIP management information, and decision management information.

1. Elements of the *agent information category* can be directly offered to a scheduling algorithm. This can either be the raw information contained by the Matrices type attribute or features contained by the Trackers field. Figure 3.7 gives an overview of the environment matrices contained by the Matrices class along with their dimensions. The system_time t and current_machine c, which are part of the decision management information, were added to exemplify how a full raw sequencing state of the GenFJS could look like.

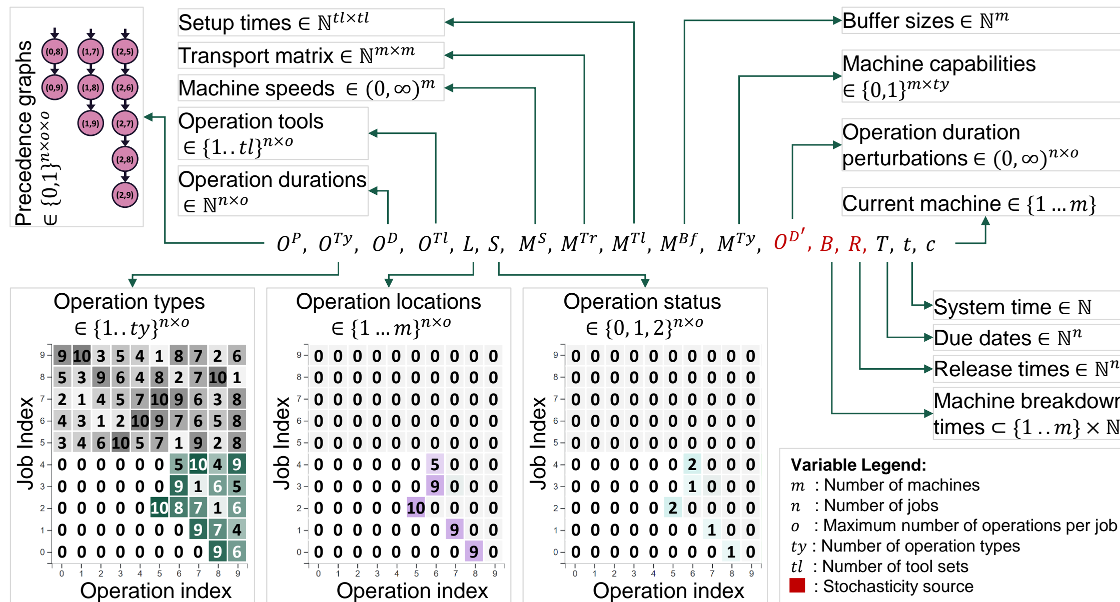


Figure 3.7: Agent information components of the environment-state and their dimensions. For concision we only exemplify four of the state components. Except for the precedence graphs, which is a stack of operation adjacency matrices, all components are at most two dimensional. RL agents should only be presented with WIP information (e.g. the green jobs in the operation type matrix). As operations finish processing, the corresponding entries in the job matrices are zeroed out. The operation location and operation status matrix track the current position and processing status of operations respectively. Red colored state components encode stochastic information and should not be presented to agents.

Note that it is not legitimate to expose the listed components in their entirety to the agent. The red variables in Figure 3.7 contain the future stochasticity, and, as such, should remain visible to the simulation only. Similarly, jobs outside the WIP should remain hidden. In the lower-left corner of the figure, we see a simplified depiction of the internal WIP representation. Green-colored job information is part of the WIP and can be shown to the agent while gray-colored information should remain obscured.

Mostly, the Matrices objects contains the tensors, matrices and vectors already listed in Section 3.2.2. However, in the state, a part of this information changes dynamically to reflect the scheduling process progress. As operations are completed, for example, the corresponding matrix entries are zeroed out, thus tracking the state progression. Additionally, the operation location matrix L tracks the position of the current job operations in terms of the machines they are currently at. The operation status matrix S tracks the processing state for operations: queued — 1, processing — 2, in transport — 0. Both location tracking matrices have a size of $n \times o$.

The main function of the Trackers object contained by the state is, as the name suggests, tracking key aspects of the simulation. The available information is listed in Figure 3.8 along with the dimension of the different attributes. The tracking information can used either directly as agent features, e.g. the total remaining work

T^{rw}, or it can be used as base feature information, e.g. the average flow time $\mu(F_j)$. The information highlighted in red in the figure contains intermediary optimization goal variables (see optimization goals — Figure 2.4).

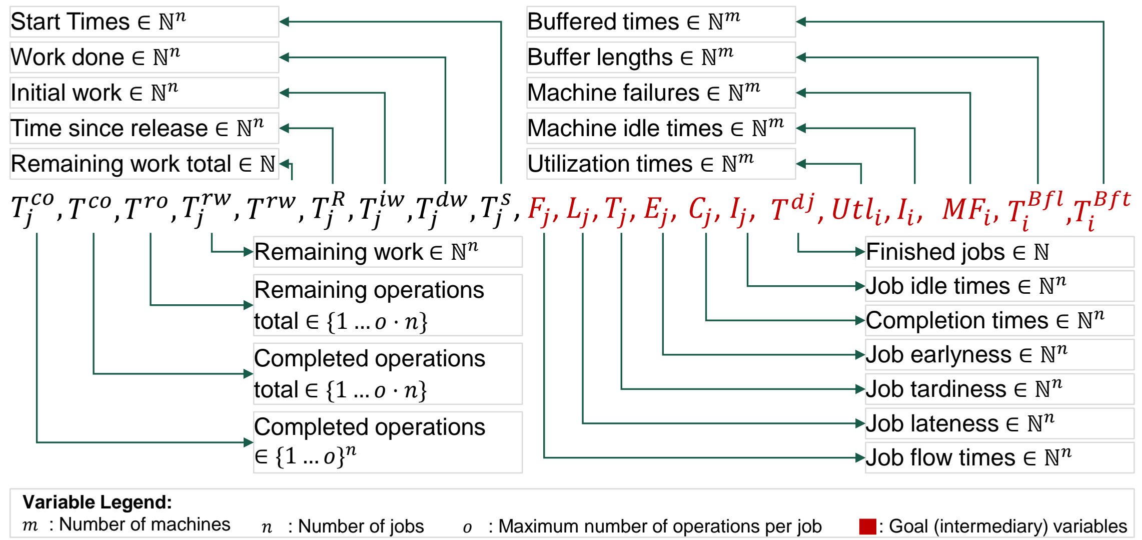


Figure 3.8: Components of the Trackers class. The red vectors contain intermediary goal variables (cmp. Figure 2.4), indexed either by the machine number, e.g. utilization times Utl_i, or intermediate or job numbers, e.g. completion times C_j. The black vectors contain additional information useful for state feature construction.

2. The second information category, namely the *convenience structures*, consists of the PrecedenceGraph and Machines state attributes. Primarily these objects allow us to retrieve legal_actions in constant time ($O(1)$). The legal action computation is not only required to allow agent to mask out illegal actions. Legal actions are essential to ensure the correct simulation run. While it would be possible to infer the legal actions from by looping through the matrices from the first state information category, that would lead to a linear asymptotic runtime relative to the number of operations $n \cdot o$ in the scheduling instance, $O(n \cdot o)$.

   The PrecedenceGraph contains the precedence DAG of all remaining operations from all jobs. The constraints corresponding to a job hang under a dummy node we call the job root, which is marked with the job number. Job roots themselves dangle under one of two global root nodes. The two roots distinguish between arrived jobs (root_visible) and jobs not yet released (root_hidden). As operations finish, they are removed from the graph. The graph allows us to retrieve the next downstream operations, and, consequently, the next downstream machines, i.e. the legal job routing actions, in constant time, $O(1)$.

   The Machines structure contains a dictionary of Machine objects indexed by the machine numbers in the system. An individual Machine type object contains a field corresponding to its input buffer which is updated throughout the simulation. As such, inferring legal sequencing actions is equivalent to looking up the current_machine within the Machines structure and retrieving its queue, which, again, takes constant time.

3. The *WIP management information*, consists of wip_jobs list, the global_to_wip dictionary, and the arrived_jobs deque. All three are properties of the state, being updated as required on every simulation step. The purpose of wip_jobs is to allow quickly retrieving the job information that an agent is allowed to see from the state matrices. Since the latter are arrays, this retrieval occurs in constant time. Conversely, global_to_wip is used to quickly find the job position within the state matrices given a particular WIP index. This is required to quickly update the target jobs in the global matrices based on the agent action, which is WIP relative. Finally, the arrived_jobs property maintains a FIFO queue of jobs that were already released, but do not fit inside the WIP. Whenever a WIP slot opens up, i.e. when all the operations of a WIP job finish, the arrived_jobs is popped and the other two attributes are updated accordingly.

4. The *decision management information* is contained by the remainder of the properties listed directly under the State class in Figure 3.2. These are the simulation_mode, system_time, current_machine, current_job, current_operation, and legal_actions properties. This information plays a vital role in switching between different state update logics on agent actions.

   The simulation mode, which can take values in $\{0, 1, 2\}$, determines the interpretation of the agent decision and, at the same time, signals the type of decision required to the scheduling algorithm. When the simulation requires a sequencing decision, the simulation mode will have a value of 0. The information about the machine for which the decision is to be taken can be retrieved from the current_machine property. When simulation_mode == 1, a job routing decision is expected. In such cases, the current_job property indicates which job is supposed to be routed. A simulation mode equal to 2 indicates job routing operations from behind a failed machine. This time, the agent does not have the freedom to chose one of several operations from a particular job (if the operation precedence allows). Instead it has to select an alternative machine for precisely one operation. The targeted operation is indicated by the current_operation property. The legal_actions property has a dual function: On the one hand, it is used by our simulation to quickly terminate the simulation if the agent tries to execute undefined actions. On the other hand, it can be used to implement the masking technique some RL algorithms can make use of (e.g. AZ; see Section 4.2.2). Finally, the system_time property is used ubiquitously throughout the simulation. Nearly all state updates, including event creation schemes require this property.

Simulation Manager: The attribute of the EventManager type contained by the core deals with handling the simulation events. More precisely, there are four queues that this objects deals with, namely the event_heap, the routing_queue, the routing_pre_q queue, and deferred_seq queue. The first queue is a binary heap and lies at the heart of the simulation. The other three objects help deal with multiple decisions needing to be taken without moving the simulation time forward, i.e. simultaneously relative to the system_time.

The event_heap consists of all the simulation events sorted ascendingly by their occurrence_time. Tie breaking is not explicitly being handled, since the order of execution for concomitant events is irrelevant within FabricatioRL, and heapq, the binary heap implementation we use, is deterministic. The events of the heap all implement the Event interface and fall into one of four categories, namely ResourceAvailability, OperationFinished, JobArrival and TransportArrival. Event objects are self-handling, the state update logic is encapsulated within the respective event. Whenever the event triggers, the handle method is called to modify the state.

Events can be either blocking, or non-blocking (blocking == false). If the event is non-blocking, the next event in the queue will be triggered upon the return of the first event's handle method. For instance, when jobs arrive, the corresponding entry is added to arrived_jobs and the tracker variables are updated. Since new job arrivals do not impact the resource availability directly, the next event triggers. Conversely, when an OperationFinishedEvent occurs, the event processing is halted upon marking the corresponding machine as free, so as to allow the agent to make the next decision.

The routing_pre queue is filled with operation indices when resources fail. In such cases all the operations buffered at the failed resource are first added to this queue. Operations from this queue are popped one by one and the scheduling agent is required to route the operations away from the failed resource. This decision loop stops when the queue is empty. Agents can alternatively chose to leave the operations blocked in the buffer behind the failed resource until the latter is repaired.

Aside from a phase immediately after initialization, the routing_queue is empty or contains exactly one operation. At the very beginning of the simulation (system_time == 0), the routing_queue contains all job indices. These indices are then popped from the queue one by one, and the agent is required to select the appropriate downstream resource. The simulation proceeds with popping the next event from the event_heap once the queue is empty. Then, whenever an operation finishes, the job to which the finished operation belonged to is added to the routing_queue, such that the sequencing decision can first be dealt with. After the sequencing decision, the simulation immediately switches to the job routing decision by popping the job index from the routing_queue. Since wait is undefined for the job routing mode, the queue empties out immediately.

The deferred_seq queue is needed to deal with the deferred sequencing decisions, as the name suggests. When the simulation is in sequencing mode, an agent can decide to defer the sequencing decision by means of a wait flag. As a result the particular resource is added to the deferred_seq queue. The simulation proceeds with asking the agent for a job routing decision and then rolls the time forward by popping the next event from the event_heap. If a new sequencing decision is required, the associated resource is first added to the deferred_seq queue. Then the simulation loops over the queue exactly once and the agent is presented with the respective sequencing decisions again. The resource indices associated with deferred sequencing decisions stay in the queue until the wait flag stops being produced by the agent or until the wait flag becomes illegal. The latter situation occurs when there are no more events to be triggered in the SimulationManager's

event_heap.

Core: The Core method names are suggestive of the link to the outer layer class (FabricatioRL). While the outer simulation layer contains the representation logic, the actual simulation logic, with the exception of the reset function, is implemented entirely by the Core class. The step method the fundamental channel of the simulation logic, functioning as the sole state-transition coordinator. Legal actions are computed and assigned to the corresponding State property by the class-private get_legal_actions method during step calls. The make_deterministic function is used by the corresponding make_deterministic_copy function of the outer layer to eliminate the stochastic elements of the state. As opposed to the FabricatioRL class, the Core is not configurable. Core.step only "understands" direct actions, i.e. operation indices for sequencing and resource indices for job destination selection decisions.

All simulation dynamics are implemented within the step method of the Core object. Figure 3.9 sketches the method's control flow as a sequence diagram. Depending on the simulation mode (1), step interprets the agent action differently and correspondingly updates the simulation state (2.1)/(2.2)/(2.3). Before returning, the method creates and queues the event resulting from the agent decision (3.1)/(3.2)/(3.3), pops a decision from the correct simultaneous decision queue (5) or triggers and handles the next events from the event heap (6)(7)(8), queues the next required decisions and changes the mode if necessary (9), and computes the next legal actions (10). The only events that are created as a result of an agent's action are of type OperationFinishedEvent and TransportArrivalEvent (renamed to ofe (3.1) and tae (3.2)(3.3) in the figure, respectively). The former is created when the simulation is in the sequencing mode ((i) in Figure 3.1), and reflects the agent's choice of operation to start processing on the machine indicated by the current_resource variable in the State. The latter event ((ii) and (iii) in Figure 3.1) is created in job routing mode and reflects the agent's dual choice of next job operation and machine that is to process the chosen operation. If the simultaneous decision queues (deferred_seq, routing_queue, and routing_pre) require handling (4), the respective event is popped from the queue (5). For simplicity, the queues are represented as the single decision_queue in the figure. If no decision needs to be been handled (4), the simulation pops events from the event_heap (7) and updates the state by calling their handle method (8) until a blocking event is encountered (6). In the final two stages, the core object then prepares the state reflecting the need for the next decision (9), and updates the legal_actions property of the state (10) before returning control to the agent. Note that the setup_decision in the figure represents a simplification analogous to decision_queue.

Requirements: The described implementation logic associated with job routing and sequencing decisions ensures the satisfaction of the *reactive breakdown coverage* requirement. Moreover, the Trackers class maintains all intermediary variables used for target (γ) computation including the current system time, thus guaranteeing *γ-traceability*.

The mechanisms of the core module described here enable an *efficient runtime*, which depends mainly on the event queue and state update procedure. During simulation

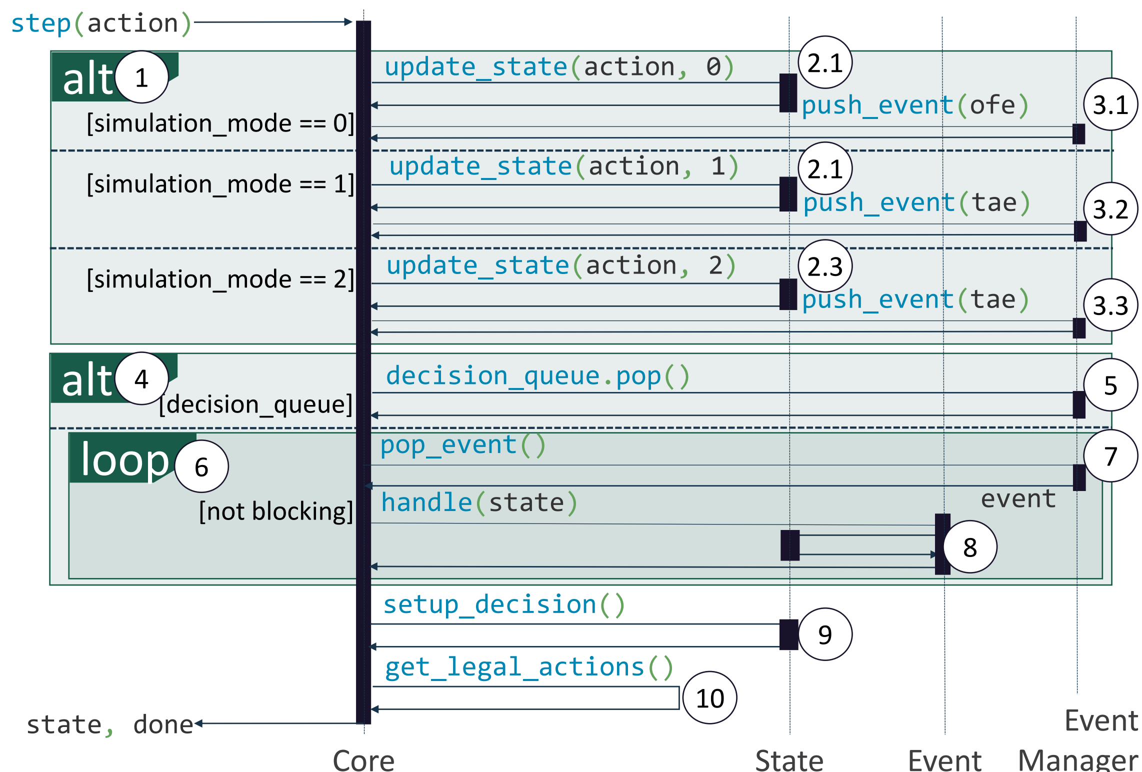


Figure 3.9: Sequence diagram of FabricatioRL's step function from the core module's point of view.

execution, events are created dynamically depending on agent decisions and stochastic influences, and added to the event_heap, which is a binary heap. The events are handled in order of their occurrence depending on their type. The event handling runtime is $O(1)$: This is because the exact positions in the State that need to be updated on occurrence are saved within the event instance on creation, e.g. the index of the operation and the machine it is processing on, for OperationFinishedEvents. Let n be the total number of operations to be scheduled during a simulation run. Every operation first needs to be transported then processed. That means that there are $O(2n)$ events to be processed. The event_heap has an insertion time of $O(\log(n))$ and $O(1)$ for popping the next event. This leads to a total asymptotic runtime of $O(2n\log(n))$.

The *planning method compatibility* requirement is enabled by the core module allowing agents to defer actions by means of a wait signal at the first cycle through the deferred_seq queue. Without the wait signal, it is not possible to iteratively build – and hence simulate – all possible schedules.

3.3 Visualization and Examples

For debugging and exemplification purposes we implemented logging functionality within FabricatioRL which works in conjunction with a visualization webapp. We detail the visualization in Section 3.3.1.

To make it easier for researchers to familiarize themselves with our code, we provide

two simple usage examples within the Github repository (Rinciog et al., 2021a) hosting our code. These examples are much simpler than the scripts associated with our later experiments which are also published. The latter are reserved for more advanced users. The introductory examples discussed in Section 3.3.2.

3.3.1 Webapp and Logger

The webapp consists of a Flask (Grinberg, 2018) backend and a pure JavaScript frontend using the D3.js (Bostock, 2022) and dagre (Newell, 2022) visualization frameworks to render the different state components. This visualization solution was chosen for its speed and flexibility. Depending on the scheduling problem, the amount of information contained in a state can be staggering. This leads to a dual problem. On the one hand, using standard python plotting libraries such as seaborn (Waskom, 2021) for the visualization of different state components can lead to long waiting times for generating the visual information. On the other hand, given the sheer amount of information contained by the scheduling state, it can become difficult to sift through the generated data in an intuitive fashion.

Visualization App: Both the backend and the frontend are conceptually straightforward in terms of architecture. The backend consists of several functions that read and parse the JavaScript Object Notation (JSON) data written by the logger and a function that compares the number of states rendered by the frontend with the number of states written by the logger in the corresponding log directory. The frontend contains a single HTML page which is constructed dynamically using JavaScript. To keep the page updated with the latest backend information, an asynchronous polling function is embedded in the page's scripts.

Every ten seconds, a call is made to the backend. If new state logs have been written, then the frontend is asynchronously updated with the new state information. Currently, the state information of a complete simulation run is loaded to the page in its entirety but kept hidden. States are then displayed one at a time using a slider.

The app's page is structured structured as can be seen in Figure 3.10. It consists of a slider at the top (1), a menu on the left (2) and a series of plots grouped into distinct tiles within the main body. Panning and zooming is set up for all visualization tiles. At the bottom of the page's main body, the contents of the event_heap are displayed as a table (3). The slider allows scrolling through the different states produced by the simulation. The system time and state number are displayed to the right of the slider (4).

There are five categories of tiles that can be displayed. The first category (5.1) displays the precedence of the remaining operations within all jobs as a graph using dagre as a visualization library. The top left tile in Figure 3.10 is an example of this (5.2).

The second category pertains to matrices (6.1). Matrix tiles (e.g. middle left) correspond to different state matrices and vectors and are visualized as heat-maps (6.2). Job-matrices, contain additional information encoded through colors. Grayed out job information corresponds to jobs currently outside the WIP. Red highlights indicate the last sequencing

Figure 3.10: Simulation visualization front-end.

decision while yellow highlights represent the other legal actions during the last decision making step.

The third category (7.1) contains Gantt charts. The toggled in tile (7.2) displays the chart of the all operations scheduled to date (e.g. middle right). Additionally, if the simulation follows a scheduling plan rather than a purely-reactive control, the plan can be logged and displayed in the front-end.

The fourth tile category contains bar plots for different goal variables, also referred to as Key

Performance Indicator (KPI). In the discussed figure, the job-centric KPI (8.1) are displayed on the bottom left tile and the resource-centric KPI (8.2) on the bottom right.

The fifth category (9.1) contains a single tile (top right) and represents the resource state using a separate table for each resource (9.2). The first table row contains the machine name and the list of types it can process. The next row contains the two headings namely "Input Buffer(b)" and "Processing". The b in the first heading represents the total number of buffered operations. The last row contains the first three operations in the buffer. The cell under the "Processing" heading contains the index of the operation currently being processed. The decision that lead to this particular state is visualized using connectors between resource tiles. Red connectors represent sequencing decisions, while green connectors represent job routing decisions.

The menu on the left (2) controls what information is displayed on the page body. In each menu group (5.1)(6.1)(7.1) check-boxes allow the user to toggle the visibility of that particular element within the body of the page. At the very bottom of the menu, the "Runs" dropdown menu (10) allows the user to switch between separate simulation runs. The log file stream corresponding to a run needs to be isolated from the others using distinct directories. Currently, the tile representing the machine state (9.1), the KPI tiles (8.1)(8.2) and the event table at the bottom (3) cannot be toggled off. For job matrices, the menu contains a radio (11) button specifying whether the displayed job matrix information should be limited to the WIP or not.

The tile set selection chosen (through checked boxed in the menu, run selection and radio button) remains in place when using the slider. As such, the user can inspect the progression of particular state aspects rather than the evolution of the state in its entirety.

Logger: To make use of the visualization, the logging functionality needs to be first turned on by specifying a non empty logfile_path string when instantiating FabricatioRL. Our simulation framework will then write log files to the specified location. If the webapp is concomitantly running, and its log input directory corresponds with FabricatioRL's log output directory, it will parse and visualizes the logfiles as they are produced by the simulation.

Turning on the logger adds considerable overhead to the simulation, hence it should not be used during runtime sensitive processes such as model training. The overhead stems from both the IO operations associated with writing files to the disk and the looping involved with piecing together the log components. For debugging purposes, we also recommend turning off any auto-play elements within FabricatioRL, such that there is less difference between two consecutive states, and, as such, the simulation logic is easier to track.

The information compiled by the logger for every state consists of eight components, which combine both State and EventManager information, and is written to the disk as a single JSON file named by the state number. The simulation data contained in the current simulation state is read by the logger and transferred into a JSON format file as

expected by the visualization app. The JSON log contains the following eight top level keys: management_info, precedence_graphs, partial_schedule, machines, matrices, pending_events, metrics_machines, and metrics_jobs. Most of this information is compiled from a single source within the State, e.g. matrices or precedence_graphs, while others are compiled using additional properties maintained by the logger, e.g. the partial_schedule, which is a Gantt chart compiled within the logger's schedule property as the simulation progresses.

3.3.2 Usage Examples

Simple Heuristic Control: Running the simulation with a simple heuristic, e.g. LPT (Benoit et al., 2021) can be done in three steps.

In a first step, the ReturnTransformer object should be implemented. LPT only operates on processing times for the buffered operations and does not consider the return signal. As such, the transform_reward function can simply return None. The transform_state function should return the legal_actions list and the operation duration matrix from the Matrices object. To be able to compute the values of different optimization goals at the end of the simulation, the Trackers object should also be returned.

The second step consists pf the setup parameter definition. Since standard JSSPs are fully described by the operation type and operation duration matrix, the other simulation parameters retain their default values.

In a third step, the simulation step function is called on a loop with an action indicated by a select_action function. The latter takes the legal_actions and operation duration matrix information and simply selects the operation from legal_actions with the least value in the the duration matrix. The desired optimization metrics can be computed from the Trackers object of the last state returned by step.

KerasRL Agent: The second scenario is a DQN training with KerasRL on randomly generated $FPOc$ instances. To showcase interface customization, we use indirect heuristic actions, (e.g. Luo, 2020), raw state information and average machine utilization as a reward. We show how to train and test the agent using seeds. This can be done in six steps.

First, the Optimizer objects for machine sequencing and job destination selection need to be implemented. Assuming these are simple priority rules, this boils down to implementing the get_action method within an Optimizer object. Said method can be implemented analogously to select_action in the LPT example above, with the distinction that the full state structure is now transparent to the method.

Secondly, the ReturnTransformer object needs to be implemented. The transform_state method takes $O^D, O^P, O^T, M^{Tr}, M^{Ty}$, together with the current machine number, current job number and the simulation mode from Matrices, flattens the multidimensional information and returns it. The transform_reward method returns the average of all the machine utilization tracker values.

Thirdly, the setup parameters including the optimizer lists and ReturnTransformer object

are used to instantiate the environment.

In the fourth step, the DQN Agent's NN architecture, is defined using Keras. The NN in- and output dimensions are obtained from the observation_space and action_space environment attributes. The NN is then passed to the constructor of the DQNAgent implemented in KerasRL.

The fifth step pertains to agent training, which is done by calling the fit method on the previously defined DQNAgent with the environment as a parameter and a number of decisions to execute before training completes. To train on a specific group of inputs, the set_seeds method should be called on the environment before the call to fit. The environment will cyclically use these seeds when re-initializing the environment on the reset background calls by the agent.

Finally, test can be called on the agent with the environment and number of episodes as parameters. A different seed set can be set for testing.

3.4 Concluding Remarks

In this section we described the steps we took towards covering the validation gap uncovered in Chapter 2 by means of an open source benchmarking simulation. To this end we first derived requirements for our framework based on the RL scheduling experiment standardization framework put forward in Chapter 2. Table 3.2 gives an overview of the requirements discussed. In terms of setup, the simulation framework should be general, extensible and γ-traceable. With respect to RL control, the framework should allow MDP configuration (breakdown, action, state, and reward), be gym compatible and be asymptotically efficient in terms of runtime. With respect to experiment validation, the framework should enable the exact reproduction of stochasticity, clearly separate inputs from control, run on traditional inputs from the literature and be compatible with planning methods.

In a next step we have shown how these requirements can be fulfilled by using a layered architecture implementing Gym. The inputs reflected by the state and separated by means of a dedicated class, allow for the coverage of many production setups. The simulation logic is centered around self-handling events and a specialised state structure allowing for an efficient runtime. The key to RL configurability is creating an interface for externally implemented objects to affect the simulation in terms of action interpretation and state and reward representation. By using RNG seeding and executing all sampling before the main simulation loop, stochasticity is made exactly reproducible.

While this work is a decisive step in the right direction for the task of validating RL approaches for production scheduling, much remains to be done in terms of framework validation, extension and actual RL benchmarking. Thorough testing of the provided framework is necessary for simulation validation.

While the simulation decribed in this section represents significant progress for the filed of RL production scheduling in particular and dynamic scheduling in general, the project

Table 3.2: Simulation Framework Requirements Derived from the Standardization Framework.

Requirement	**RL Scheduling Aspect**	**Requirement Type**
Generality	Scheduling Setup	Functional
γ-Traceability	Scheduling Setup	Functional
Modularity	Scheduling Setup	Non-Functional
Reactive Breakdown Coverage	RL Design	Functional
Action-Space Configurability	RL Design	Functional
State-Space Configurability	RL Design	Functional
Reward Configurability	RL Design	Functional
Extended Gym Compatibility	RL Design	Functional
Runtime Efficiency	RL Design	Non-Functional
Reproducible Stochasticity	Validation	Non-Functional
Input Separation	Validation	Functional
Backwards Compatibility	Validation	Functional
Planning Compatibility	Validation	Functional

should by no means end here.

More development effort should be spent along four lines. First, our simulation framework should be extended to reach a 100% coverage of the scheduling problems in literature. Secondly, the simulation should be more extensively tested, re-factored and documented. Note that code makes use of type-hints, documents the main simulation functions and provides a 70% code line coverage through tests. Still, more can be done, particularly with respect to testing the more fringe setups made possible by our simulation. Thirdly, the visualization, which is now, for now, only prototypical, could be developed into a production level app. This could help make it easier for new researchers to the field to get started with FabricatioRL in particular and dynamic production scheduling in general.

Chapter 4

From Theory to Implementation: Selected RL Scheduling Algorithms

The way positive reinforcement is carried out is more important than the amount

— B. F. Skinner

RL approaches position themselves somewhere between exact and priority rule approaches. As with exact re-planning solution approaches, RL schedulers make decisions in the current state while taking future conditions into account. RL schedulers are similar to priority rules in that they can react adaptively to changes in the production state (Waschneck et al., 2018).

Note that "adaptivity" is not a well defined term. For terminological disambiguation, we define "adaptivity" in the context of scheduling as the capability of an algorithm to retain its solution quality under uncertain/shifting environment conditions. In general, an algorithm is said to be adaptive if "changes its behavior based on the information available at the time it is running" (Montillet et al., 2016). Estivill-Castro et al. (1992) define adaptive (sorting) algorithms as those capable of taking advantage of the order within their inputs, i.e. smoothly growing functions of both the input disorder. For Zaknich, 2005 adaptive (filter) algorithms are those seeking "to minimize an appropriate objective or error function that involves input, reference and [...] output signals". Adaptive schedulers are defined by Ferm et al. (2010) as algorithms that "change [their] scheduling scheme according to the recent history and/or current behavior of the system". Our definition is in line with both the idea of reacting to environment conditions put forward by Estivill-Castro et al. (1992), Zaknich (2005), Ferm et al. (2010), and Montillet et al. (2016) and the optimization included by Zaknich (2005).

In stochastic environments, RL solutions may be preferable to re-planning using exact approaches and extensive search for three reasons. First, a priori planning assuming deterministic inputs may be thwarted by stochastic events occurring during production, such as resource availability issues or new job arrivals. Secondly, finding optimal or near-

Table 4.1: Perceived RL solution approach advantages compared to exact re-planning solutions and priority rules.

Criterion	**Priority Rules**	**RL**	**Exact Re-Planning**
Deterministic Setup Solution Quality	Medium	High	Optimal
Adaptivity	High	High	Low
Transferability	High	High	Variable
Runtime Efficiency	High	High	Low
Mathematical Modeling Overhead	Low	Low	High

optimal solutions for the planning problem is computationally taxing for large production instances. Finally, such solutions require exact mathematical descriptions of the problem at hand, which are sometimes difficult to formulate for complex setups (e.g Rinciog et al., 2020).

As opposed to simple priority rules, RL approaches could learn to leverage patterns in the scheduling problem, leading to better scheduling solutions e.g. Kuhnle et al., 2020. The priority rule approach disregards any structure that may be inherent to the given problem, i.e. priority rules are myopic (Luo, 2020), and is, by design, not optimal. On the flip side, priority rules could still work well in uncertain environments (Wang et al., 2017) and require no expensive computation.

From the RL method comparison with exact approaches and simple priority rules, the following potential RL advantages become apparent. First, RL solutions are seen as adaptive (e.g. Hu et al., 2020b), being potentially more robust in stochastic environments than exact approaches (e.g. Wang et al., 2021c) and yielding better quality solutions than priority rules (e.g. Luo, 2020). Secondly, RL models, once trained, are efficient in terms of runtime (e.g. Waschneck et al., 2018). Thirdly, the mathematical modeling overhead of RL approaches is low e.g. Baer et al., 2019.

Table 4.1 succintly sums up these postulated advantages. Both priority rules and RL are expected to be highly adaptive, albeit on a different solution quality level. A different aspect of adaptivity is captured by the transfer learning capability associated with RL. In the scheduling context, transfer learning implied that models trained on particular instances or problems can be successfully deployed to different instances or problems with little or no loss of solution quality. Since neither priority rules nor exact approaches involve learning, we capture this aspect under the "transferability" attribute. Priority rules are highly transferable. The transferability of exact approaches depends on the problem and the particular type of exact re-planning solution approach. In terms of runtime, we consider polynomial solutions as highly efficient and exponential solutions as inefficient.

Motivated by the presented RL advantages, in this Section, we formally introduce the

two RL algorithms, namely DDQN and AZ, which we will later employ as production scheduling solvers. We start by laying down the mathematical RL formalism required for the detailed understanding our chosen algorithms in Section 4.1. Thereafter, we motivate our specific algorithm choice (DDQN and AZ) and describe their particularities from theory to implementation in Section 4.2.

4.1 Preliminaries

Recall the broad description of the RL loop from Section 2.3: Agents take actions within an environment based on the current state. The environment takes the agent action and transitions into a next state while providing the agent with a reward signal. It is by means of this rewards, that agents, which seek to maximize their future reward, can improve their action selection strategy.

This interaction is captured by the MDP formalism which we formally introduce in Section 4.1.1 along with the Generalized Policy Iteration (GPI) which abstractly describes the way in which agents improve. Based on the mathematical MDP formalization, we then detail the exact way in which agents can achieve their reward maximization goal within in different RL approach paradigms. Value-based approaches represent the foundation for most RL theory and are introduced in Section 4.1.2. The value-based complement, namely the policy-based approach, is introduced in Section 4.1.3. The brief Section 4.1.4 elucidates the main idea behind the actor-critic paradigm.

The information in this section can be found in a different form in varied RL manuals, albeit in a less concise fashion. We heavily rely on the seminal work of Sutton et al. (2018), who developed an introduction manual that was refined and adapted over more than 20 years (e.g. Sutton et al., 1998) to lay the theoretical foundation that follows. If not otherwise indicated, the reader is to assume that the lain down concepts are extracted from the Work of Sutton et al. (2018).

4.1.1 Markov Decision Process and the Generalized Policy Iteration

MDP are formally described as the quadruple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$. $\mathcal{S}$ is the designation of the state-space, $\mathcal{A}$ represents the action-space, $\mathcal{R}$ defines the set of rewards and $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is the so called dynamics function. The dynamics function gives the probability of transitioning from some state $s \in \mathcal{S}$ into another state $s' \in \mathcal{S}$ and receiving a reward $r \in \mathcal{R}$ on taking an action $a \in \mathcal{A}$, as described by Equation 4.1. Here, and in all subsequent equations, S_t, R_t and a_t denote state, rewards and actions at a particular time t.

$$p(s', r \mid s, a) := \mathbb{P}(S_t = s', R_t = r \mid S_{t-1} = s, a_{t-1} = a). \tag{4.1}$$

MDP are sometimes described as the quintuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, r)$. While the dynamics function fully defines the decision process, it is often split into a state-transition function

$p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ and a reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ [1]. These two functions are defined for the general case by equations 4.2 and 4.3 respectively, where t denotes the current time and S_t, R_t the respective state and reward values at time t.

$$p(s' \mid s, a) := \mathbb{P}(S_t = s' \mid S_{t-1} = s, a_{t-1} = a) = \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \tag{4.2}$$

$$r(s, a) := \mathbb{E}[R_t \mid S_{t-1} = s, a_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) \tag{4.3}$$

Note that for *deterministic* RL environments, given a current state s and a fixed action a, $p(s' \mid s, a) = 1$ for exactly one next state $s' \in \mathcal{S}$ and 0 for all other combinations. For instance, when making a move a from a position s in chess, the state that follows, s', is a certainty defined by the game rules. Hence, assuming fixed prior states s and actions a, the transition probability is 1 for the triple (s', s, a) only. All other board configurations $s'' \neq s'$ have a probability $p(s''|s, a)$ of 0. In stochastic environments, however, the same does not hold.

The decision process lain down by $(\mathcal{A}, \mathcal{S}, \mathcal{R}, p, r)$ has the Markov property, meaning that the transition probability from one state to another is only dependent on the current state-action pair and not on any of the previous, i.e.

$$\begin{aligned} &\mathbb{P}(S_t = s \mid (S_{t-1} = s_n, a_{t-1} = a_n), \dots, (S_{t_0} = S_0, a_{t_0} = a_0)) \\ &\qquad = \mathbb{P}(S_t = s \mid S_{t-1} = s_n, a_{t-1} = a_n). \end{aligned} \tag{4.4}$$

The agent's job is to maximize the value of the *expected return* denoted as G_t, where t is the current time-step. Whenever the agent-environment interaction is finite, i.e. after T time steps there is no more action that needs to be taken, G_t can be defined as sum of rewards $R_{t+k}, 1 < k < T - 1$. To avoid G_t becoming ∞, the rewards are generally discounted using a parameter $\gamma \in [0, 1)$:

$$G_t := \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + G_{t+1} \tag{4.5}$$

RL agents use a policy function π to decide which action to take in any given state. π maps state-action pairs to the probability of the particular action yielding the most reward, i.e. the action $\text{argmax}_a \pi(a \mid s)$ should be chosen from given state s. The policy is tightly bound to the concept of state- and action-value, $v_\pi(s)$ and $q_\pi(s, a)$. These functions can be used to evaluate how "good" a particular position s is, and how "good" an action a taken from a position s is respectively. Both value functions are dependent on the agent policy π and represent an approximation of the expected future reward. Formally, $q_\pi(s, a)$ and

[1] Using p for both the transition function and the dynamics function constitutes abuse of notation, but is the norm as per Sutton et al. (2018).

$v_\pi(s)$ can be defined as per Equations 4.6 and 4.7:

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \sum_{a' \in \mathcal{A}} \pi(a' \mid s') \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s', A_{t+1} = a'] \\
&= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \sum_{a' \in \mathcal{A}} \pi(a' \mid s') q_\pi(s',a') \qquad (4.6) \\
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s)(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']) \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s)(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) v_\pi(s')) \qquad (4.7)
\end{aligned}
$$

The optimal action- and state-value functions are defined as the return of the *optimal* policy π_* for every state (Equations 4.8, 4.9). While optimal value functions are impossible to compute for most real-world RL applications, the relationship defined by the two equations also is the same for *good* policies and value functions. This implies that, provided we find a good approximation of either value function, we can construct a good policy and vice-versa.

$$
\begin{aligned}
q_*(s,a) &:= \max_\pi q_\pi(s,a) =: q_{\pi_*}(s,a), \forall s \in \mathcal{S} \qquad (4.8) \\
v_*(s) &:= \max_\pi v_\pi(s) =: v_{\pi_*}(s), \forall s \in \mathcal{S} \qquad (4.9)
\end{aligned}
$$

This idea is reflected by a generic technique called GPI. Hereby, an agent starts off with a random policy, which is evaluated through environment interaction by means of computing (an approximation of) the associated value function (e.g. q). This stage is called policy evaluation. The policy improvement stage ensues thereafter. During this stage, the policy is changed by greedily following the value function estimate to collect as much reward as possible. These two steps are repeated until the policy and value functions stop improving (see Figure 4.1). Note the GPI loop lain down is an abstraction. The granularity and interdependence of the two phases is conditional on the particular RL algorithm.

There are three main classes of RL algorithms to choose from implementing GPI, namely value-based (v), policy-based (π), and actor-critic approaches (π, v). Value-based methods learn an approximation of a value function first and define the policy implicitly through the value function. Conversely, policy-based methods learn a policy directly. Actor-critic methods combine both approaches by learning the policy directly but employing a value function approximation to inform the learning process.

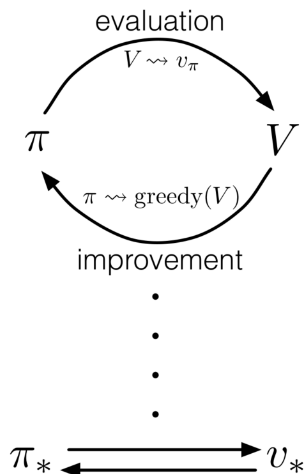


Figure 4.1: Generalized Policy Iteration GPI. Policy π and value function estimation V evolve through alternating evaluation and improvement phases until, ideally, an optimal policy π_* or value v_* function is found. Optimal policies induce optimal policy functions and vice-versa. (Source: Sutton et al., 2018)

4.1.2 Value-Based RL Approaches

The prototypical value-based approach is the so called Temporal Difference Learning TD(λ). The algorithm works as follows. First, the state value function approximation $V : \mathcal{S} \rightarrow \mathbb{R}$ is initialized to random values. For each step of an episode, i.e. the complete collection of interactions until the environment encounters a terminal state, the agent first selects an action a as per policy π induced by V. After observing the reward R_t and the next state S_{t+1}, the agent computes the state-value error at time t as per Equation 4.10 and uses it to correct the value function during update as per Equation 4.11.

$$\delta_t := R_t + \gamma V(S_{t+1}) - V(S_t) \tag{4.10}$$

$$V(S) \leftarrow V(S) + \alpha \delta_t e_t(S), \forall S \in \mathcal{S} \tag{4.11}$$

The update rule in Equation 4.11 above uses so-called eligibility traces e_t are defined recursively by $e_0(S) = 0, \forall S \in \mathcal{S}$ and $e_t(S) = \lambda\gamma e_{t-1}(S) + \mathbb{1}_{\{S_t=S\}}, \forall S \in \mathcal{S}$.[2] The eligibility trace mechanism simply tracks the visited states, such that the temporal difference error δ can be used to update more than one state at a time, albeit in a discounted fashion. λ is the eligibility trace discount factor. The α parameter in Equation 4.11 is called the learning rate.

The most straightforward way of representing V would be a simple table of values that indexes all environment-states $S \in \mathcal{S}$ directly. This, however, is infeasible for large state-spaces such as those involved in production scheduling, depending on the respective

[2]Here and henceforth, $\mathbb{1}$ is an indicator function. Formally, indicator functions $\mathbb{1}_A$ are defined over a base space Ω and take values in $\{0, 1\}$. $A \subseteq \Omega$ is an event. $\mathbb{1}_A$ takes the value 1 for arguments ω, if and only if $\omega \in A$. The notation $\{X = k\}$ defines the event wherein a stochastic variable X takes the value k (cmp. Henze et al., 2010; Durrett, 2019).

state-space modeling. Particularly continuous state components cannot be stored in a table without quantization, which potentially leads to loss of information thereby impeding algorithm convergence, or leading to a local optimum. Neural networks (NN) $f_\theta : \mathcal{S} \to \mathbb{R}$, as universal function approximators (Csáji, 2001), could be used instead. The policy evaluation step becomes a simple lookahead search to select the action leading to the state s' with the highest value $f_\theta(s')$ starting from the current state s.

Equations 4.10 and 4.11 become, 4.12 and 4.14. Here, we generalized the notation used by Sutton et al. (2018) by involving the NN loss function l, e.g. Mean Squared Error (MSE), explicitly in the computation of the state value error term $J_t(\theta)$. The loss function $l : \mathbb{R} \times \mathbb{R} \leftarrow \mathbb{R}$ computes an error term $l(y, \hat{y})$ between an expected/target value y and an estimated value $\hat{y}$. Within the current context, the target value is given by the sum of the current reward R_t and the estimated future reward of the next state $f_\theta(S_{t+1})$. The estimation is represented by the neural network's predicted value of the current state $f_\theta(S_t)$. Note that when computing the gradient of the error term $l(R_t + f_\theta(S_{t+1}), f_\theta(S_t))$ with respect to network weights θ, the numeric value of the first argument is used, while the network function is expanded in the second argument.

$$J_t(\theta) := l(R_t + f_\theta(S_{t+1}), f_\theta(S_t)) \tag{4.12}$$

$$e_t \leftarrow \lambda\gamma e_{t-1} + \nabla_\theta J_t(\theta), e_0 = 0 \tag{4.13}$$

$$\theta \leftarrow \theta + \alpha J_t e_t \tag{4.14}$$

The eligibility trace discount factor λ can take any value between 0 and 1. Setting λ to 0 and deferring the lookahead search in favor of the action-value estimation function Q yields two popular algorithms, namely SARSA and QL. The error and value estimation update equations for both SARSA and QL (4.15 and 4.16) are, as one would expect, very similar to TD(λ). The eligibility trace term could be updated for Q values as $e_t(S, A) := \lambda\gamma e_{t-1}(S, A) + \mathbb{1}_{\{S_t=S, A_t=A\}}, \forall(S, A) \in \mathcal{S} \times \mathcal{A}$, but since λ is 0, $e_t(S)$ is one for exactly the current state S_t and otherwise 0.

$$\delta := R_t + \gamma \max_{a_{t+1}} Q(S_{t+1}, a_{t+1}) - Q(S_t, a_t) \tag{4.15}$$

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha\delta \tag{4.16}$$

The difference between the SARSA and QL is given by the *moment when the action for* S_{t+1} *is chosen*, not by the error term computation or value function update scheme. SARSA evaluates the policy to choose actions for both S_t and S_{t+1} before updating the Q function (and thus the policy), while QL chooses the action for S_t, updates the policy, and then chooses the value for S_{t+1}. This results in the policy of the previous step not necessarily

being followed, which is why QL is said to be *off-policy*.

Again, it is possible to use NN, this time $f_\theta : \mathcal{S} \rightarrow \mathbb{R}^{|\mathcal{A}|}$ to represent Q. The network weights are then updated as per Equation 4.18. In Equation 4.17, the error term $J(\theta)$ only changes at index a_t of the previously chosen action $a_t = \mathrm{argmax}_a f_\theta(S_t)$ and l is once again the NN's loss function.

$$J(\theta)_{a_t} = l(R_t + \gamma \max_a f_\theta(S_{t+1}), f_\theta(S_t)_{a_t}) \tag{4.17}$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)_{a_t} \tag{4.18}$$

4.1.3 Policy-Based RL Approaches

An alternative to the value-based methods described up to this point is given by the so-called Policy Gradient (PG) methods. Herein a parameterized policy π_θ is used to select actions directly. While PG methods may still employ a value function to learn the policy parameters, the former need not be consulted when selecting an action. PG methods define an objective function $J(\theta)$ as the expected reward under policy π_θ starting from the beginning of the agent-environment interaction. The objective function is initialized to the value of the first state $v_\pi(S_0)$ under the policy π (Equation 4.19). The policy is then improved iteratively by computing its gradient with respect to the policy weights (Equation 4.21) and updating these using gradient ascent (Equation 4.22).

$$J(\theta_0) := v_{\pi_\theta}(S_0) \tag{4.19}$$

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{S \in \mathcal{S}} \mu_{\pi_\theta}(S) \sum_{a \in \mathcal{A}} \pi_\theta(a|S) q_{\pi_\theta}(S, a) \tag{4.20}$$

$$\begin{aligned} &\overset{\text{PG Theorem}}{\propto} \sum_{S \in \mathcal{S}} \mu_{\pi_\theta}(S) \sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(a \mid S) q_{\pi_\theta}(s, a) \\ &= \mathbb{E}_\pi[q_\pi(S, a) \nabla_\theta \ln \pi_\theta(a \mid S)] \\ &= \mathbb{E}_\pi[G_t \nabla_\theta \ln \pi_\theta(a_t \mid S_t)] \end{aligned} \tag{4.21}$$

$$\theta_{t+1} := \theta_t + \alpha \nabla_\theta J(\theta_t) \tag{4.22}$$

Computing the objective gradient is made possible by the policy gradient theorem. The indirect dependence of the stationary distribution $\mu_{\pi_\theta}(S)$ of the Markov Chain on the policy π_θ together with the direct dependence of the action probabilities $\pi_\theta(a \mid S)$ on the policy π_θ (first line of Equation 4.21) makes computing the partial derivatives with respect to θ difficult. Because of the policy gradient theorem, it is possible to compute a value proportional to the objective function gradient by only calculating the partial derivatives of π_θ with respect to its weights θ (second line of Equation 4.21). The objective function gradient can now be expressed as the expectation under policy π_θ of the product of the expected return G_t and the gradient of the natural logarithm of the policy π_θ with respect to θ. We refer the reader to Sutton et al. (2018) for the proof of the PG theorem.

Given the expectation representation of the gradient, there are quite a few ways to approximate the PG. The REINFORCE approach, for instance, involves the Monte Carlo approximation of the policy-dependent expectation in the last line of Equation 4.21. To this end, trajectories, i.e. sequences of state, action, rewards (S_1, a_1, R_1), (S_2, a_2, R_2), $\dots, (S_T, a_T, R_T)$ are generated with the current policy π_θ. Using the sum of immediate rewards $\sum_{i>t} R_i$ observed along the trajectory as an approximation of the expected return G_t at time point t, we can then update the policy weights as per Equation 4.23, for every trajectory point $t \in \{1, \dots, T\}$:

$$\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln\pi_\theta(a_t \mid S_t) \tag{4.23}$$

4.1.4 Actor-Critic RL Approaches

Actor-critic approaches combine policy- and value-approaches into one. Having already introduced these two paradigms, the description of actor-critic systems can be kept (very) brief.

Instead of using environment interaction to approximate the expected future reward G_t, as in the REINFORCE example, actor-critic approaches use a value function approximator, i.e. the critic, to inform the policy approximator, i.e. the actor, of the quality of its action. The action value function $Q(S, a)$ could be trained separately following a value-based approach, e.g. DQN. Its evaluations $Q(S_t, a_t)$ of actions a_t given states S_t at time point t is then used directly within the PG computation replacing G_t in Equation 4.21, leading to Equation 4.24. The expectation term is approximated by gathering a large number n of $(S_t, Q(S_t, a_t), \pi_\theta(a_i|S_i))$ tuples, and averaging out the corresponding terms $Q(S_i, a_i)\nabla_\theta \ln\pi_\theta(a_i \mid S_i)$.

$$\begin{aligned}\nabla_\theta J(\theta) &\propto \mathbb{E}_\pi[G_t \nabla_\theta \ln\pi_\theta(a \mid S)] \\ &\approx \frac{1}{n}\sum_{i=1}^{n} Q(S_i, a_i)\nabla_\theta \ln\pi_\theta(a_i \mid S_i)]\end{aligned} \tag{4.24}$$

$$\theta_{t+1} := \theta_t + \alpha\nabla_\theta J(\theta_t) \tag{4.25}$$

4.2 Selected RL Algorithms

Section 2.3.3 has shown the great diversity of employed RL algorithms within the field of scheduling. From the many available algorithm options we chose AZ and a flavor of DQN, namely DDQN, as our RL scheduling champions.

The value-based DDQN algorithm was selected because of its relative simplicity and the overwhelming popularity of its predecessors QL and DQN within the RL production

scheduling community (see Figure 2.11). The DDQN flavor of QL was selected as it is state-of-the art within the family alongside DDDQN, and is implemented by Stable-Baselines 3 (Raffin et al., 2021), an actively maintained RL agent library. Conversely, the actor-critic AZ approach was chosen because of its innovative combination of different RL and search paradigms and its few applications within the field of production scheduling.

We describe the inner-workings of DDQN and AZ in Sections 4.2.1 and 4.2.2 respectively.

4.2.1 Double Deep-Q Networks

The DDQN algorithm introduced by Van Hasselt et al. (2016) distinguishes itself from the generic QL algorithm in three ways, which we describe in the following paragraphs. First, a NN is used as a value function approximator. Secondly, a technique called memory-replay is used during training to curtail overfitting and increase data efficiency. Thirdly, a target network is introduced to improve learning stability and reduce target value overestimation, hence the name "Double DQN". The first two extensions are shared by its predecessor DQN, which was first described by Mnih et al. (2015). We conclude this section with a brief discussion of the exploration vs exploitation dilemma and an implementation overview.

NN Approximator: In the case of DQN, a neural network $f_\theta : \mathcal{S} \rightarrow \mathbb{R}^{|\mathcal{A}|}$ is used to represent Q. Given a state S_t agents choose the action with the highest value as predicted by the network (Equation 4.26). The error term is calculated as per Equation 4.17. Large values of the discount factor γ serve to shift focus from the immediate reward to future rewards. In Equation 4.18, which describes the network update, the error term $J(\theta)$ is non-zero only at index a_t of the previously chosen action $a_t = \text{argmax}_a f_\theta(S_t)$ and l represents the NN's loss function. Parameter α defines the DQN learning rate. In supervised learning terms, the "ground truth" vector $\hat{y}$, which has a dimension of $|\mathcal{A}|$, is identical to the predicted vector $y := f_\theta(S_t)$ save for the position a_t.

$$a_t = \underset{a}{\text{argmax}} f_\theta(S_t) \tag{4.26}$$

Memory Replay: More sophisticated DQN agents (e.g. Pilchau et al., 2020) do not perform error term computation and NN updates using a *single* experience tuple (S_t, a_t, R_t, S_{t+1}) which results from taking action a_t in state S_t. Instead, the agent uses a limited size FIFO queue D named "replay memory" to store experience tuples whenever actions are taken. The FIFO nature of the queue serves to reflect a preference for more recent experiences, associated with a better policy over old experiences. The neural network is trained on targets constructed as per Equation 4.17 using the experience from a random minibatch drawn from D after a predetermined number of steps. Sampling mini-batches helps avoid training on data that is too similar. This is because consecutive states and the corresponding error terms tend to be highly correlated. Additionally, a higher data efficiency is achieved because of the repeated use of the same data points during training.

Target Network: DDQN builds upon DQN by introducing a second network into the training process seeking to improve network stability and reduce target value over-estimation. DDQN separates between the online network f_θ used for action selection and the target network $f_{\hat{\theta}}$ used to compute the target values for the error term. When updating the online network (Equation 4.18) $f_{\hat{\theta}}$ is used instead of f_θ in the first argument of the loss function in Equation 4.17. The two networks are different solely with respect to the value of their weights. Every k steps, the weights θ of the online network are used to update the target network weights $\hat{\theta}$. A soft target weight update, as proposed by Lillicrap et al. (2015), using an additional parameter $\tau \in (0, 1]$, can be performed (Equation 4.27) instead of directly copying the network weights.

$$\hat{\theta} = \tau\theta + (1 - \tau)\hat{\theta}, \theta \in (0, 1). \tag{4.27}$$

Exploration vs exploitation: DQN agents use an ϵ-greedy strategy to tackle the exploration versus exploitation dilemma (Sutton et al., 1998). Exploitation, i.e. using the knowledge encoded by the agent NN to choose actions, is needed in order to maximize reward gains. Exploration, i.e. ensuring that the state-space is sufficiently inspected, facilitates a better approximation of the optimal Agent policy. When using the ϵ-greedy strategy, the agent chooses a random action with probability $\epsilon \in (0, 1)$. The initial values of epsilon, i.e. ϵ_{max} is multiplicatively decayed each time an action is chosen using the $\epsilon_{\text{decay}} \in (0, 1)$ parameter. The value of ϵ used during training can be capped as specified by ϵ_{min}.

Implementation Overview: Figure 4.2 visualizes the algorithm at a glance and Table 4.2 presents the corresponding algorithm parameters. For concision we deferred visualizing the ϵ-greedy action selection scheme. We also neglected the visualization of the frequency of the online network update (every $k_{\text{self-play}}$ steps), and, similarly the target network update frequency (every k_{target} steps).

During the self-play stage, the online network f_θ takes agent-states S_t^A and produces the action values $\hat{y}$ from which the action a_t corresponding to the maximum value is chosen and passed along to the environment, moving it into the next state S_{t+1}^E. The reward produced by the environment R_t on transition is saved within the replay buffer D alongside the old agent-state, the action that lead to it, and the new agent-state as the experience tuple $(S_t^A, a_t, R_t, S_{t+1}^A)$. Every $k_{\text{self-play}}$ steps, a mini-batch b is sampled from D, and the error term is computed for every sample i using the experience tuple. To this end, the MSE between the output vector $\hat{y} := f_\theta(S_t^A)$ and the target vector y is calculated. The vector y and $\hat{y}$ are identical, save for the value at the position a_t, which, for y corresponds to $R_i + \gamma \max_a f_{\hat{\theta}}(S_{i+1}^A)$. f_θ is updated using the error term $l(y, \hat{y})$ (Equation 4.17) using the learning-rate α (Equation 4.18). Every k_{target} steps, the $f_{\hat{\theta}}$ is additionally updated as per Equation 4.27.

The above elaboration is sufficient for readers to understand the parametrization of the Stable-Baselines 3 (Raffin et al., 2021) implementation which we use during our experiments. This agent library was chosen because of its continued support, implementation speed

Table 4.2: DDQN parameters at a glance.

Name	Symbol	Description
Self-Play/Target Network	$f_\theta / f_{\hat{\theta}}$	Self-Play/Target Neural Network taking states as inputs and outputting the value of all posible actions in the given state.
Learning Rate	α	The rate at which the Self-Play Network is updated given the error term gradient (see Equation 4.18).
Buffer Size	β	The maximum number of experiences in the replay buffer.
Batch Size	b	The number of experiences to sample from the replay buffer.
Polyak Update Coefficient	τ	The soft update parameter used when updating the target network.
Discount factor	γ	The future reward prioritization factor (see Equation 4.17).
Initial Exploration Rate	ϵ	The initial probability of random actions.
ϵ-Decay	ϵ_{decay}	The exploration rate decrease factor.
Minimum Exploration Rate	ϵ_{min}	The lower bound of the exploration rate.
Training Frequency	$k_{\text{self-play}}$	The number of steps after which a self-play network update is performed.
Target Update Interval	k_{target}	The number of steps after which a target network update is performed.

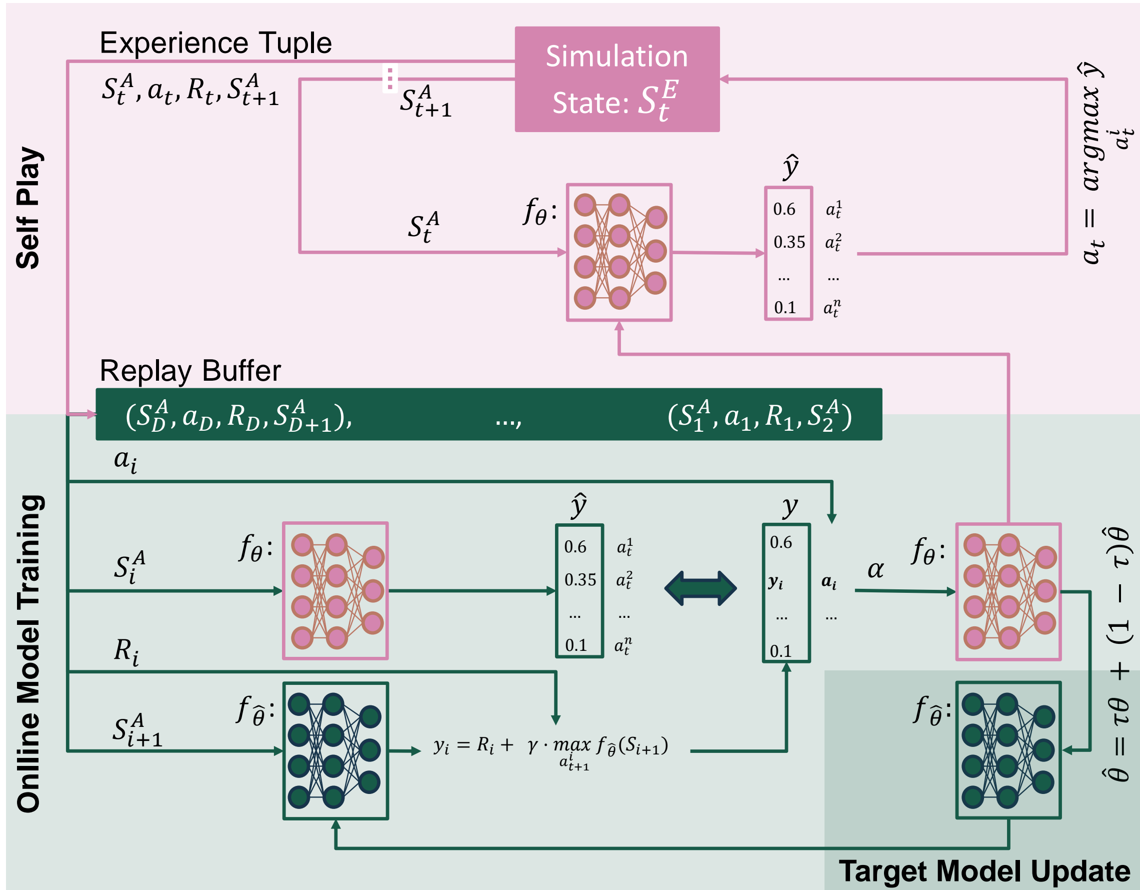


Figure 4.2: Double DQN Self-play and training process.

and diversity of implemented RL agents, including PPO, A2C and TD(λ) employed by the production scheduling literature. As such, our experiment scripts can function as a template for easily testing alternative agent algorithms. The implementation speed stems from the fact that Torch (Collobert et al., 2011), a low level NN framework, is used with all the implemented agents. Additionally, Stable-Baselines 3 offers the possibility for monitoring the agent training process using TensorBoard (Abadi et al., 2015). Note that while the library does not list DDQN among the implemented algorithms, the listed DQN is in fact a DDQN.

4.2.2 AlphaGo Zero

While originally designed for perfect information (board-)games, AZ's range of applications has since expanded to the realm of combinatorial optimization with two use cases for scheduling (Rinciog et al., 2020; Yu et al., 2020) being listed in Chapter 2. More recently, more generic applications such as using AZ for search-based solutions to the Boolean Satisfiability Problem (SAT) have surfaced (Dantsin et al., 2022).

We proceed to discuss AZ taking the following six aspects into account. First, we contextualize the algorithm and offer some terminological disambiguation helping to isolate the AZ version we employ. Secondly, we succinctly present the NN approximator used by the algorithm. Thirdly, we illustrate the algorithm's action selection mechanism, which relies on MCTS. Fourthly we outline the AZ training process. Fifthly, we explicate

the concept of masking. Sixthly, we report the details of our implementation. We conclude this elaboration with a brief consideration of the RL class AZ falls under.

Disambiguation: There are four versions of the AZ, namely AlphaGo Lee (Silver et al., 2016), AlphaGo Zero (Silver et al., 2017b), AlphaZero (Silver et al., 2017a) and MuZero (Schrittwieser et al., 2020). As opposed to the DQN family, the second version of the algorithm is much simpler than the first. The difference between the second and the third version is marginal, the third version representing a generalization of the second which does not use board symmetries present in the game of Go. The last version, which is capable of mastering board-games without being endowed with knowledge of the game rules, is in its incipient phase having been just recently published. We describe and use a variation of the second and third versions here and refer to it generically as AZ.

AZ (Silver et al., 2017b), follows a previous version, namely AlphaGo Lee (Silver et al., 2016), and was proven to outperform its predecessor. The new algorithm, while building upon results from the previous work, by using a combination of NN and MCTS, radically simplifies both the training and the evaluation process. This is being done by introducing four main changes. First of all, the algorithm is now trained solely through self-play, without the use of human data. Secondly, the network input was changed to consider raw states only, with no additional features. Thirdly, the authors merged the previously distinct policy and value networks into one with two output heads. Last but not least the MCTS was simplified.

NN Approximator: The NN, f_θ with neuron weights θ, at the core of AZ combines policy and value function into one. f_θ takes a stack of k consecutive raw state representations, with the current state S_t at the top, as its input and outputs the pair $(\boldsymbol{p}, v) := f_\theta(S_t, \dots, S_{t-k}) =: f_\theta(\boldsymbol{S_t})$ of action probabilities $\boldsymbol{p}$ and the expected game result v from the position S_t. Note that variables in bold represent vectors. $\boldsymbol{p}$ and v are produced by two dedicated output heads stacked on top of a ResNet architecture (He et al., 2016) with 40 residual layers followed by a final convolutional layer. The stack of residual layers act as feature extractor, pre-processing the information in $\boldsymbol{S_t}$ before feeding it into the sub-networks associated with the two heads, which are referred to as the "policy head" and "value head". The output heads are independent of each other in the sense that the gradient contribution of the policy-head does not impact the gradient in the value-head network and vice-versa. However, the gradient of the feature extractor weights depends on both heads. Note that, for simplicity, we drop the time indices t in what follows.

Action Selection: The AZ action selection strategy relies on MCTS. The central structure of MCTS is a game tree with nodes representing states $S \in \mathcal{S}$ and directed edges representing actions $a \in \mathcal{A}$ possible from the particular state. Each edge in the game tree stores the probability of being the best move $P(S, a)$, a visit count $N(S, a)$ and the average expected result $Q(S, a)$. Nodes store the expected result $V(S)$. The MCTS algorithm consists of three phases, namely selection (1), expansion and evaluation (2), and backpropagation (3), that are executed several times before every action. The execution of these steps leads to the tree structure growing dynamically, as showcased in Figure 4.3:

1. During the MCTS *selection phase* nodes reached over edges maximizing

$$Q(S,a) + P(S,a) \cdot c_{\text{puct}} \frac{\sqrt{\sum_{a'} N(S,a')}}{1 + N(S,a)} \propto \frac{P(S,a)}{1 + N(S,a)} \tag{4.28}$$

   are recursively selected until a "dangling edge" (an edge pointing to no node) is encountered. Note that edges are added to the tree before the nodes they lead to, hence the dangling edges. In the equation above, c_{puct} is a tunable exploration parameter and $\sum_{a'} N(S,a')$ is the cumulative visit count of all node outgoing edges.

2. In the *expansion and evaluation phase*, the new leaf node S_L^n reachable over the selected dangling edge is first added to the tree. We use the indices n to denote the current MCTS iteration and L to indicate that the node is a leaf. In a second step, f_θ is used to compute the best action probabilities and node value $(\boldsymbol{p}, v) = f_\theta(S_L^n)$. Then $V(S_L^n) := v$ is added to the freshly created node and the node's outgoing edges are added to the tree and associated with the corresponding values of network's probability vector $P(S_L^n, \cdot) := \boldsymbol{p}$. If the freshly added node is terminal (i.e. corresponds to an end-game state), the node value is collected from the environment rather than from f_θ and no edges are added.

3. The value $V(S_L^n)$ is additionally used to update all edges upwards on the selection path in the *backpropagation phase*. The update implies incrementing the edge visit counts $N(S,a)$ and setting their values $Q(S,a)$ to the average accumulated value up to the current iteration. Using the indicator function $\mathbb{1}_{\{S,a,i\}}$ to denote that edge a of node S was selected during the ith MCTS iteration, Equation 4.29 describes these value updates.

$$Q(S,a) = \frac{1}{N(S,a)} \sum_{i=1}^{n} \mathbb{1}_{\{S,a,i\}} V(S_L^i) \tag{4.29}$$

After the predetermined number of MCTS iterations the final move selection is performed. AZ introduces a temperature parameter $\tau \in (0,1]$ to this step. The visit counts of the root edges are raised to the power of $1/\tau$, and normalized through division by $\sum_a N(S_{\text{root}}, a)^{1/\tau}$. This induces a probability distribution over the legal actions relative to the root node. The final move is then selected by sampling from this distribution. Therefore, τ controls the degree of exploration during RL. If τ is equal to 1, there is a higher chance that an action not corresponding to the edge with the highest visit count will be selected. When set to an infinitesimal value, the chance that an edge other than that with the most visit counts is selected becomes negligible.

Training: AZ has a streamline network training process. AZ training has three steps, which are repeated until the model weights saturate. These are self-play (i), neural-network training (ii) and model selection (iii):

(i) During the *self-play* stage, which is depicted in Figure 4.4a, the MCTS scheme

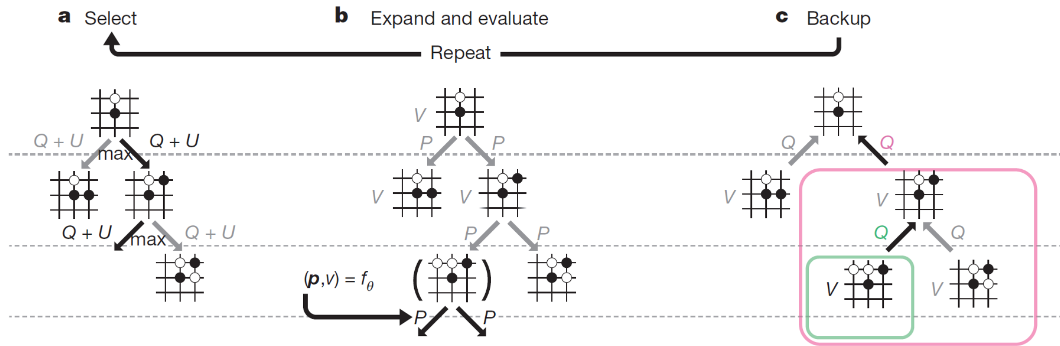


Figure 4.3: The MCTS process in AlphaGo Zero: (a) Nodes are selected recursively by traversing edges corresponding to an action $a = \text{argmax}_a Q(S, a) + U(S, a)$ until the outgoing edge of a leaf node is reached (traversed edges are indicated by black arrows). (b) A new node s_L is added to the tree, $(P(S, \cdot), V(S_L)) = f_\theta(S_L)$ is evaluated and its outgoing edges are updated with probabilities $P(S, \cdot)$. (c) Action values Q are updated up the tree path using the mean of all state values V stored in the nodes. Source: Silver et al., 2017b, AZ MCTS

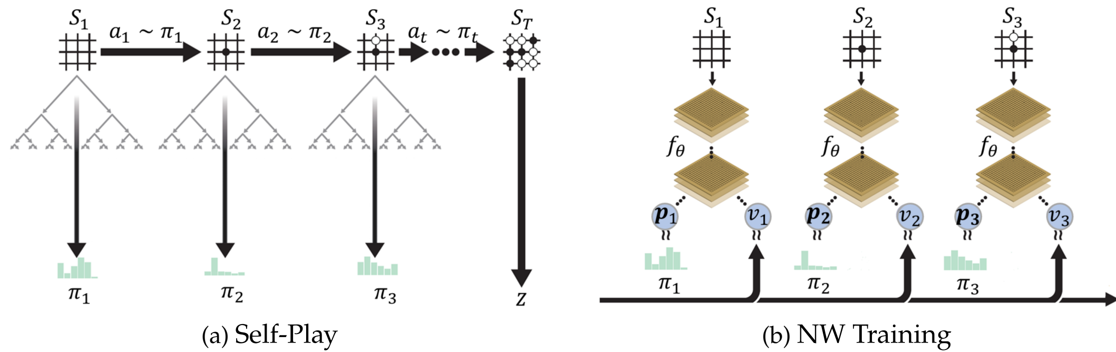
introduced above is used to play games from start to finish, i.e. iterations 1 to t, by sampling from probability distributions $\boldsymbol{\pi_i}$ returned by the MCTS algorithm for states s_i. Board positions and the corresponding MCTS action probabilities $(S_i, \boldsymbol{\pi_i})$ are stored for every performed move. When a terminal state S_T is reached, the reward $z := r(S_T)$ is used to form triples $(S_i, \boldsymbol{\pi_i}, z)$.

(ii) The AZ network can now be *trained* on them using gradient descent to minimize the loss function in Equation 4.30 where $(\mathbf{p}, v) = f_\theta(S)$ and c is a parameter for the $L2$ weight regularization. The loss function l combines the MSE loss for the value head (regression) and the Categorial Cross Entropy loss for the policy head (classification). The training process is depicted in Figure 4.4b.

$$l = (z - v)^2 - \boldsymbol{\pi}^T \log \mathbf{p} + c||\theta||^2 \tag{4.30}$$

(iii) Finally, to make sure data is being generated using the best model, the post training *model is pitted against the old version of itself* that generated the data. If the new version wins, the model used for data generation (i.e. self-play) is replaced by the new version. In non adversarial settings, the models before and after training could be evaluated by simply using the cumulative reward, e.g. the game score, as a criterion.

Masking: In AZ, agents are said to "know the game rules", i.e. no illegal action is ever taken. This is achived by using a technique called masking: The network output determining the next moves is multiplied by a bit mask with ones corresponding to legal actions given a particular state. The result is then re-normalized to represent a probability distribution through division by the masked vector sum. To do the same for our scheduling agents, we simply query the environment for legal actions and compute a mask from it. This mask we then pointwise multiply with the network's output.



(a) Self-Play (b) NW Training

Figure 4.4: (a) Games are played from start to finish, using neural MCTS to select a move. The move probabilities $\boldsymbol{\pi_i}$ are stored together with the corresponding state s_i. When the end-state S_T is reached, the reward z is associated with every pair $(s_i, \boldsymbol{\pi_i})$. (b) The network f_θ is trained using gradient descent to minimize a combination of mean squared error on the value head (pairs (S_i, z)) and cross-entropy on the policy head (pairs $(S_i, \boldsymbol{\pi_i})$). (cmp. Silver et al., 2017b).

Implementation Overview: To better use the particularities of production scheduling problems, we made two slight adaptations to the original algorithm described above. First, instead of collecting the reward in the terminal state only, we collect the immediate reward $z_t := r(S_t)$ at every intermediate decision time-point t. This increase in feedback should induce more stability in the training process, since the likelyhood of radically different states being associated with the same reward is decreased. Secondly, we eliminate the model comparison pitting stage. This is because our preliminary experiments have shown this step to have no significant impact on the model performance, while significantly increasing the training time.

Table 4.3 gives an overview of the parameters associated with the steps discussed. By comparing the AZ parameters with those of DDQN, we see a first potential advantage of the former approach: Because of the smaller number of parameters, the tuning overhead should be smaller for AZ. Aside from the MCTS parameters, AZ has no additional parameters when compared to supervised learning NN.

We implemented AZ from scratch in Python using tensorflow.keras (Abadi et al., 2015), a high level NN API, for the policy-value network. We chose keras because of its intuitive API which helped eliminating implementation overhead. The MCTS implementation is recursive and operates on a tree structure represented as a collection of dictionaries with state representations (nodes) as keys. A different deterministic copy of the simulation is used to explore future states and evaluate them for every iteration of the tree search procedure. TensorBoard is used for training visualization.

Note on RL Class: While AZ falls loosely into the actor-critic RL class, it in fact defies all standard RL categories. The AZ policy π is comprised of a probabilistic search algorithm, namely MCTS, embedding a policy function P, and a (state) value function V. Both functions are implemented using a NN. Similarly to actor-critic approaches V is used to inform the policy which ultimately chooses the action in any given state. However, unlike typical actor-critic approaches, the value function is not directly used during gradient

Table 4.3: AZ parameters at a glance.

Name	Symbol	Description
Policy-Value Network	f_θ	Neural Network with two different output stacks taking states as inputs and returning the probability of the best action (policy-head) and the values associated with the current state (value-head).
Learning Rate	α	The rate at which the network is updated given the error term gradient (see Equation 4.22).
Buffer Size	β	The maximum number of experiences in the replay buffer.
Number MCTS Iterations	i_{max}	Determines the number of times the MCTS stages selection (1), expansion (2), rollout (3) and back-propagation (4) are executed before choosing the root node action.
Puct Paramer	c_{puct}	MCTS exploration parameter influencing the weight of node visit counts during the search.
Temperature	τ	Exploration parameter increasing or decreasing the chance that the highest probability action is selected after MCTS.

computation. Unlike typical value-based approaches, V is not trained using any estimation of future reward (e.g. Q in Equation 4.15). Instead, V is updated using solely the reward signal. Finally, unlike policy approaches (Equation 4.21) the PG is not computed over the policy (which combines MCTS, P and V functions) in its entirety, rather the policy network P alone is updated as per Equation 4.21. Though this categorial ambiguity makes formal theoretical analyses of AZ difficult, its results are quite compelling.

Chapter 5

Leveling the Playing Field: RL-Competitive Scheduling Baselines

If you don't set a baseline standard for what you'll accept in life, you'll find it's easy to slip into behaviors and attitudes or a quality of life that's far below what you deserve

— Anthony Robbins

As delineated in Section 2.5, RL is not the only class of algorithms positioned between replanning using exact approaches and heuristic solutions. In this Section, we seek to move towards covering the RL baselining gap by describing competitive baseline algorithms that share the qualities of RL approaches. Recalling the Table 4.1 from Chapter 4, this entails designing production scheduling solution approaches that offer a high solution quality in deterministic setups (1), are highly adaptive (2), efficient in terms of runtime (3) and require little modeling overhead (4).

All models are designed to interact with the production simulation framework FabricationRL (see Chapter 3). The MDP breakdown broadly defining this interaction is that of Interlaced Routing and Sequencing. Hereby, whenever a resource m_i becomes free, two decisions have to be taken in sequence. First, an operation index has to be chosen from the ones available for processing in the resource buffer. Secondly, if job route alternatives are available, a machine index needs to be selected for the downstream processing of the job.

We distinguish between baselines that do not require a simulation for decision making and call them "snapshot-based" approaches, from "simulation-based" baselines that require an optimization simulation. For approaches in the snapshot-based category, the only information required for decision making is a static representation of the current production state. Conversely, simulation-based approaches require the full dynamics of a production simulation to reach a decision. Note that this constitutes a fair comparison with RL methods, since these fall in the second category as well. Why this is the case should be fairly obvious for AZ: The simulation is directly employed during the MCTS stage. DDQN,

along all other model free RL approaches, also fall in this category since they require the full production simulation for training. The simulation made available for training, could be modified to load a current shop-floor snapshot such that it can also be used for decision making, which is why AZ and our simulation-based baselines constitute adequate algorithmic sparring partners. Note however, that only a few Manufacturing Execution System (MES) (Ugarte et al., 2009) in place today have the live monitoring capabilities necessary for the provision of an accurate snapshot state to initialize the simulation.[1]

Three of the four baselines we elaborate on are not domain independent. While, RL domain independence is often used as an argument for the approach, this property of RL does not hold independently of RL design. All designs employing (domain dependent) indirect actions, e.g. priority rules, become domain specific by extension. Similarly, the domain independence of RL approaches relying on the Iterative Search Refinement or Iterative Re-Planning breakdown are domain specific since the underlying search procedures are domain dependent. Given the large body of work employing domain specific RL, we do not include domain independence in our RL-competitiveness definition, and do not require our baselines posses this property.

Free of the domain independence requirement, we design our baselines for production scheduling setups including both sequencing and job routing decisions. Specifically, we ensure that our solutions can be tested on $(FJc|r_j^s, M_i^o|C_{\max})$ setups and all the subsumed problems. Note, however, that the simulation-based approaches can be used as solvers for sequencing and decision problems in any setup.

We describe the snapshot-based baselines in Section 5.1. Section 5.2 introduces our simulation-based baselines. This chapter is concluded by several remarks on the RL-competitiveness of our baselines in Section 5.3.

5.1 Snapshot-Based Approaches

The snapshot-based approaches we discuss here are inspired by the classical OR production scheduling literature and consist of priority rules approaches and a constraint programming approach. We introduce the two baseline categories in Section 5.1.1 and Section 5.1.2 respectively. The priority rules approaches are chosen as baselines because they are ubiquitous in the scheduling literature (both RL and otherwise) and surprisingly effective depending on the production scheduling problem instance at hand. Furthermore they represent the starting point for a significant volume of RL work, more specifically the RL experiments using them as (indirect) actions (see Section 2.3.2).

5.1.1 Simple Heuristics

Priority rules, as the name suggests, define priorities for operations or machines depending on whether the rules are meant to be utilized for sequencing on machines or job destination

[1]This is supported by an unpublished meta-analysis of MES providers (Kuball, 2022) using the market analysis works put forward by Wiendahl et al., 2021 and MES-D.A.CH-Verband, 2021 as primary literature. The study suggests that, albeit small, an intersection set of MES providers and scheduling setups wherein such shop-floor snapshots are available is likely to already exist.

decisions (see Figure 3.1). Given a sequencing decision request from a resource m with operations $O_{Bf} := \{(j_1, i_1), \ldots, (j_k, i_k)\}$ waiting in its buffer, sequencing priority rules first sort the buffered operation using specific criteria. The highest priority operation, i.e. the operation at the head of the sorted queue, then gets assigned to the requesting resource. Job routing priority rules work analogously. Given job with several downstream machine alternatives for the next operation, job routing priority rules sort the resources in question by some priority criterion and select the head of the sorted list as the decision.

Concept: The sorting criteria we consider are operation processing time (a), remaining processing time in the job associated with the operation (b), the number of remaining operations in the job of provenance (c), the ratio of remaining processing time to the number of remaining operation in the job of provenance (d), the due date associated with the job of provenance (e) and the total load of the machines that can be assigned to the remaining associated job operations (f).

Sorting descendingly by criteria (a) to (d) gives rise to the LPT, Longest Remaining Processing Time (LRPT), Most Operations Remaining (MOR) and Most Time Per Operation (MTPO) priority rules. Conversely, sorting ascendingy by criteria (a) to (d) yields the SPT, Shortest Remaining Processing Time (SRPT), Least Operations Remaining (LOR), and Least Time Per Operation (LTPO) heuristics. For criteria (e) and (f) we only sort ascendingly. The associated heuristics are EDD and Least Utilized Downstream Machine (LUDM). It should be noted that LUDM is a heuristic developed by us. The intuition behind it is that we should prefer operations that would lead to a more balanced load of machines further down the line.

Job routing heuristics, which are additionally required when creating priority rule solutions for scheduling environments with job routing flexibility, work analogously. Given a job routing request for job j upon the conclusion of one of its operations, simple job routing heuristics prioritize the possible next downstream machines based on some criteria and pick the highest priority resource as the decision.

In this work we employ two different job routing criteria leading to four simple heuristics. The criteria are the number of operations buffered at the next eligible machines (i) and the amount of processing time buffered at the next eligible resources (ii). Sorting descendingly by (i) and (ii) results in the Most Queued Operations (MQO) and Most Queued Time (MQT) heuristics respectively. Sorting ascendingly defines the complementary Least Queued Operations (LQO) and Least Queued Time (LQT).

Table 5.1 gives an overview of the employed baseline heuristics. Aside from the associated decision, sorting criteria, number, name, abbreviation, and ordering, we additionally provide the FabricatioRL `State` variables required for implementing them in the "Variables Needed" column. The "Variables Needed" column refers to either tracker variables listed in Figure 3.8 or raw state information depicted in Figure 3.7. The heuristic list used in this work is certainly not exhaustive. There are many more discussed in literature (e.g. Panwalkar et al., 1977).

Implementation: FabricatioRL makes implementing and testing priority rule approaches

Table 5.1: Overview of the simple heuristic baselines.

Decision	Sorting Criteria	Variables Needed	Number	Name	Abbre-viation	Ordering
Sequencing on Machines	(a) Operation duration	O^D_{ji}	1	Shortest Processing Time	SPT	↗
			2	Longest Processing Time	LPT	↘
	(b) Total remaining processing time of the containing job	T^{rw}_j	3	Shortest Remaining Processing Time	SRPT	↗
			4	Longest Remaining Processing Time	LRPT	↘
	(c) Number of remaining operations within the containing job	T^{ro}_j	5	Least Operations Remaining	LOR	↗
			6	Most Operations Remaining	MOR	↘
	(d) Ratio between remaining operations and remaining processing times within containing job	T^{rw}_j, T^{ro}_j	7	Least Time Per Operation	LTPO	↗
			8	Most Time Per Operation	MTPO	↘
	(e) Due date of the containing job	d_j	9	Earliest Due Date	EDD	↗
	(f) Total load of downstream machines of the containing job	O^{Ty}, M^{Ty}, T^{Bft}_i	10	Least Utilized Downstream Machines	LUDM	↗
Job Destination Selection	(i) Sum of queued processing times	T^{Bft}_i	1	Least Queue Time	LQT	↗
			2	Most Queued Time	MQT	↘
	(ii) Number of queued operations	T^{Bfo}_i	3	Least Queued Operations	LQO	↗
			4	Most Queued Operations	MQO	↘

easy. The heuristic implementation boils down to first retrieving the operations/machines currently in need of prioritization by means of a call to get_legal_actions or reading the corresponding property directly from core.state. All the information required for the sorting criteria is also maintained by the State structure. Our scripts contain dedicated classes which implement the Optimizer interface for each heuristic (see Figure 3.2). Rather than sorting the n operations/resources in need of prioritization, which would run in $O(n \cdot \log(n))$, we simply extract the highest priority by calculating a minimum/maximum per specified criteria in a single pass. This takes linear time.

The implementation of LUDM follows this same principle, though its implementation is slightly more convoluted. Mathematically our heuristic can be expressed as in Equation 5.1. Herein $\mathcal{M}$ is the index set of all machines, o is the maximum number of operations per job. The sum in the equation is computed for all jobs j associated with buffered operations

at the current machine, a set we dubbed J_{Bf}. The sum itself considers all the machine m-operation i, combinations. If the operation i of job j is unfinished ($\mathbb{1}_{\{O_{ji}^{Ty} \neq 0\}} \neq 0$) and machine m process the operation's type, i.e. O_{ji}^{Ty}, which is indicated by the the entry 1 in the operation type matrix, i.e. $M^{Ty}_{mO_{ji}^{Ty}} = 1$, then the load of the machine m contributes to the sum. Otherwise the addend is 0.

$$j = \underset{j \in J_{Bf}}{\text{argmin}} \sum_{m \in \mathcal{M}, i < o} \mathbb{1}_{\{O_{ji}^{Ty} \neq 0\}} \cdot \mathbb{1}_{\{M^{Ty}_{mO_{ji}^{Ty}} = 1\}} \cdot T_m^{Bft} \tag{5.1}$$

The implementation of LUDM, while slower than that of other heuristics, is faster than suggested by the above formula. This is because, given the operation index contained by the legal actions retrieved from the environment, we can retrieve all machines capable of executing the operation in constant time from a machine capability dictionary maintained by the state.matrices object. The machine indices can then be used to retrieve each of the buffer loads from the state.trackers.buffer_times in constant time The heuristic is also slightly slower since after retrieving the remaining operations associated with the job the operation candidate is a part of we need to retrieve the all machines capable of executing the operation and sum up the associated loads.

The priority rules implementation can be both passed to the environment for indirect action designs or be used as external controls using the HeuristicControl class which implements the Control interface. The latter only defines one method, namely play_game which takes a simulation instance in an initial state as a parameter and returns the simulation end state to the caller. HeuristicControl uses the heuristic objects it receives during initialization to step through the simulation until the end state is reached. Depending on the simulation_mode, get_action is called either on the job routing or sequencing heuristic to retrieve the appropriate step parameter.

5.1.2 Constraint Programming Heuristic

Solving a problem with CP amounts to mathematically formulating the constraints defining the problem. The mathematical formulation can then be almost directly plugged into a solver. In the following set of mathematical models, we use bold math, to indicate decision variables. Decision variables are to be set by the model. Conversely, parameters, which we use in conjunction with decision variables to define model constraints, are fixed.

Concept: We formulate a CP model to solve all setups subsumed by the flexible job-shop with machine operation capabilities and makespan as an optimization goal ($FJc|M_i^o|C_{\max}$) subsequently. We start by defining the simpler job-shop setup Jm and build up to the FJc and $FJc \mid M_i^o$. After introducing the optimization goal $C_{\max}$, we describe the formalism allowing for the model deployment in setups subsumed by ($FJc \mid r_j^s, M_i^o \mid C_{\max}$). The formal definitions are concluded by the definition of the operations subsets characterizing our CP heuristic.

The input to a Jm consists of n jobs and m resources (or machines). All jobs consist of

exactly m distinct operations that can be mapped to precisely one machine. Hence, the type o^{ty}_{ji} of operation i from job j is uniquely identified by the machine m_{ji} it can be processed on: $o^{ty}_{ji} = m_{ji}$. All operations types within a job are distinct with respect to their type. Each operation o_{ji} has an associated duration o^{d}_{ji}. Any operation $i + 1$ within a job can only be executed if the previous operation i has been completed beforehand (precedence constraints — Equation 5.2). Each machine can only process one operation at a time (no overlap constraints — Equation 5.3), and once an operation has started, it must be executed without interruption for its whole duration (no preemption — Equation 5.4). Assuming that $\mathcal{J}$ is the index set of all jobs and O_j the index set of the operations from job j, o^{ty}_{ji} the type of the ith operation from job j, $\boldsymbol{s_{ji}}$ the operation start time and C_{ji} the operation completion time, the above can be expressed as:

$$i_1 < i_2 \Rightarrow \boldsymbol{s_{ji_1}} < \boldsymbol{s_{ji_2}} \qquad \forall j \in \mathcal{J} : \forall i_1, i_2 \in O_j \tag{5.2}$$

$$m_{j_1 i_1} = m_{j_2 i_2} \Rightarrow \boldsymbol{s_{j_1 i_1}} + o^{d}_{j_1 i_1} \leq \boldsymbol{s_{j_2 i_2}} \vee \boldsymbol{s_{j_2 i_2}} + o^{d}_{j_2 i_2} \leq \boldsymbol{s_{j_1 i_1}} \qquad \forall j_1, j_2 \in \mathcal{J} : \forall i_1, i_2 \in O_{j_1} \times O_{j_2} \tag{5.3}$$

$$C_{ji} = \boldsymbol{s_{ji}} + o^{d}_{ji} \qquad \forall j \in \mathcal{J} : \forall i \in O_j \tag{5.4}$$

The FJc extends the Jm setup by introducing routing alternatives (Equation 5.6). While in the Jm case the mapping of an operation type to a machine was unique, in the FJc case, as defined by Pinedo (2012), an operation can be executed on one of several machines from exactly one of $c \leq m$ non-overlapping work centers C_i (Equation 5.5). This means, that an operation type o^{ty}_{ji} is represented by its corresponding work-center index $c_{ji} \in \{1, \ldots, c\}$. Each job consists of c operations of distinct types. The machine assignment $\boldsymbol{m_{ji}}$ of operation o_{ji} is now a decision variable:

$$o^{ty}_{j_1 i_1} \neq o^{ty}_{j_2 i_2} \Rightarrow C_{o^{ty}_{j_1 i_1}} \cap C_{o^{ty}_{j_2 i_2}} = \emptyset \qquad \forall j_1, j_2 \in \mathcal{J} : \forall i_1, i_2 \in O_{j1} \times O_{j2} \tag{5.5}$$

$$\boldsymbol{m_{ji}} \in C_{c_{ji}} \qquad \forall j \in \mathcal{J} : \forall i \in O_j \tag{5.6}$$

The β-parameter M^{o}_{i} eliminates the non-overlapping property of the work centers. Additionally, the operation types within a job need not be unique anymore. To distinguish between the $(FJc|)$ and $(FJc|M^{o}_{i})$ we drop the work-center assignment variables c_{ji} and use the operation type parameter $o^{ty}_{ji} \in \{1, \ldots, ty\}$ directly henceforth. We use set variables $\mathcal{M}_i$ to denote the subset of resources that can process operations of type i which leads to the following expression:

$$\boldsymbol{m_{ji}} \in \mathcal{M}_{o^{ty}_{ji}} \qquad \forall j \in \mathcal{J} : \forall i \in O_j \tag{5.7}$$

The definition of the makespan $C_{\max}$ optimization goal in Equation 5.8 completes the scheduling problem definition. Makespan is defined as the completion time of the last

operation in the system. The goal of a scheduling algorithm is to set the operation start times $\boldsymbol{s_{ji}}$ and, in the case of FJc, the operation machine assignments $\boldsymbol{m_{ji}}$, such that the makespan is minimized. Using $\mathcal{M}$ to denote the index set of all machines, this can be expressed as

$$\min_{\boldsymbol{s_{ji}},\boldsymbol{m_{ji}} \in \mathbb{N}\times\mathcal{M}} C_{\max} := \min_{\boldsymbol{s_{ji}},\boldsymbol{m_{ji}} \in \mathbb{N}\times\mathcal{M}} \max_{j,i \in \mathcal{J}\times O_j} C_{ji} = \min_{\boldsymbol{s_{ji}},\boldsymbol{m_{ji}} \in \mathbb{N}\times\mathcal{M}} \max_{j,i \in \mathcal{J}\times O_j} \boldsymbol{s_{ji}} + o^{d}_{ji} \tag{5.8}$$

The $(FJc \mid M^o_i \mid C_{\max})$ model, being more general, also applies to the more specific FJc and Jm setups. The sole difference between the two being given by the contents of the machine alternative sets $\mathcal{M}_{o^{ty}_{ji}}$ for operations i from jobs j of type o^{ty}_{ji}. In the Jm case, $\mathcal{M}_{o^{ty}_{ji}}$ has a cardinality of one irrespective of operation index ij. In the FJc case, machine alternative sets have a cardinality greater than one for at least some operation types, but these sets do not overlap for different types, i.e. $\forall j_1, j_2 \in \mathcal{J} : \forall i_1, i_2 \in O_{j1} \times O_{j2} : \mathcal{M}_{o^{ty}_{j_1 i_1}} \cap \mathcal{M}_{o^{ty}_{j_1 i_1}} = \emptyset$ (compare the non-overlapping work centers equation — Equation 5.5).

To use this model in the dynamic setups, the CP solver needs to be redeployed to construct a schedule using time dependent operation sets O^t_j and job sets $\mathcal{J}_t$. These sets are computed at the time t of new job arrivals. The job operation sets O^t_j (Equation 5.9) contain the indices of operations from job j whose start times, as computed by the previous iteration of the CP solver, lie in the future, $\boldsymbol{s_{ji}} > t$, or were not yet added to the model, $\boldsymbol{s_{ji}} = \perp$. The $\perp$ value helps distinguish between decision variables contained by the model that need to be set anew ($\boldsymbol{s_{ji}} > t$) and decision variables that need to to be added to the model, e.g. those corresponding to operations from new job arrivals. We define $\mathcal{J}_t$ (Equation 5.10) as the set of job indices corresponding to the not yet finished jobs with a release time smaller than t.

$$O^t_j := \{i \in O_j : \boldsymbol{s_{ji}} \geq t \vee \boldsymbol{s_{ji}} = \perp\} \tag{5.9}$$

$$\mathcal{J}_t := \{j \in \mathcal{J} : r^s_j \leq t \wedge |O^t_j| \neq 0\} \tag{5.10}$$

Re-planning is often disregarded as a baseline, because of the long computation times associated with solving larger instances of NP-hard problems such as the setups at hand. We circumvent the runtime drawback by limiting/clipping the number of operations considered for planning to at most three per job (CP3) as well as limiting the search time allotted to the CP solver to 100 seconds. Given our WIP setup, we additionally only consider the jobs $\mathcal{J}^{WIP_t}$ within the WIP at timepoint t. We define the new operation index sets in Equation 5.11 (clipped operation sets), thereby completing the model employed.

$$\begin{aligned} O^{WIP_t 3}_j := \{&i, \ldots, i+l : \\ &((\boldsymbol{s_{j(i-1)}} < t \wedge (\boldsymbol{s_{ji}} >= t \vee \boldsymbol{s_{ji}} = \perp) \wedge i \neq 0) \vee \\ &(\boldsymbol{s_{ji}} >= t \wedge i = 0)) \wedge l = \min(3, |O^t_j|)\} \end{aligned} \tag{5.11}$$

Implementation: Having computed an initial schedule, we look it up whenever the environment requests a sequencing or job routing decision respectively. Algorithm 1 sketches the interaction between the FabricatioRL instance sim and the scheduling module implementing the described model cp3. The schedule can be seen as a Gantt chart indexed by resource numbers with values corresponding to a start time sorted list of job, operation index pairs. If a sequencing action is required for the resource m (line 5), we simply pop the first operation from the corresponding schedule list (line 6). If a job routing decision is required (line 18), we check the job index j of every first position in the schedule until j matches the job index indicated by the simulation state (lines 12 to 17). We then return the corresponding schedule key, i.e. the machine index.

Algorithm 1 Scheduling with CP3 and FabricatioRL

Input: sim, cp3
Output: state

```
 1: state ← sim.reset()
 2: schedule ← cp3.solve(state)
 3: done, wip_j ← false, state.wip
 4: while ¬ done do
 5:     if state.mode = 0 then                                ▷ Sequencing
 6:         a ← schedule[state.m].pop(0)
 7:         if a ∉ state.legal ∨ a = ⊥ then
 8:             schedule ← cp3.solve(state)
 9:         end if
10:         a ← schedule[state.m].pop(0)
11:     else
12:         m ← 0
13:         while state.j ≠ schedule[m][0][0] do
14:             m ← m + 1
15:             continue
16:         end while
17:         a ← m
18:     end if                                ▷ Job routing; state.mode = 0
19:     state, _, done, _ ← sim.step(action)
20:     if state.wip ≠ wip_j then
21:         schedule ← cp3.solve(state)
22:         wip_j ← state.wip
23:     end if
24: end while
```

The schedule needs to be recomputed before selecting an action based on it in three situations. First, by design, the schedule is mostly incomplete with respect to all the operations in WIP. As such situations may arise, when no operation is found in the schedule on a sequencing decision request ($a = \perp$ in line 7). Secondly, the operation indicated by the schedule may not yet be present in the buffer at the requesting resource (indicated by the operation index not being contained by state.legal_actions in line 7). In such situations, we should technically output a wait signal to the simulation. However, since CP3 has only limited information, we chose to ignore potentially costly wait signals. Instead, the

schedule is recomputed. Finally, the schedule is recomputed when the wip index set changes (line 20).

We use the ORTools (Perron et al., 2022) CP SAT solver in conjunction with FabricatioRL for our implementation. There are three classes involved in the solution implementation namely CPPlanner, ReplanningCPSequencing and CPControl.

The CPPlanner is responsible for interfacing with the ORTools library. On initialization, the planner first gathers the relevant variables from the FabricatioRL state, e.g. operation durations, machine alternatives, and adds them to the ORTools sat.python.cp_model.CpModel model together with the FJc constraints decribed in this section. The $\boldsymbol{s_{ji}}$ and $\boldsymbol{m_{ji}}$ decision variables are then assigned within an ORTools sat.python.cp_model.CpSolver object by calling Solve on it with the CpModel as a parameter. After generating the solution, the planner object populates the machine_to_assigned_tasks dictionary indexed by machines with values corresponding to the sequence of operations mapped to it in the order of their starting times. This is achieved by reading the $\boldsymbol{s_{ji}}$ and $\boldsymbol{m_{ji}}$ values from the CpSolver object post Solve call. Sequencing decision are retrieved from the dictionary during call to choose_fixed_plan_action. If job routing decisions are required, as indicated by the state parameter passed to the latter method, the appropriate $\boldsymbol{m_{ji}}$ variable is read directly from the solver object.

The ReplanningCPSequencing class, which uses our CPPlanner object, implements FabricatioRL's Optimizer interface and manages the re-scheduling logic described by the inner loop of Algorithm 1. Finally, the CPControl object encapsulates the interaction between the ReplanningCPSequencing optimizer and the FabricatioRL object within the play_game function.

5.2 Simulation-Based Approaches

The simulation-based approaches combine the indirect action idea encountered in RL literature with the future reward estimation principle. Similarly to RL the indirect action expected to maximize the future reward is taken in any given state. However, the simulation is used directly instead of relying on prediction.

These two methods we describe in here, namely SimSearch (Section 5.2.1) and MCTS (Section 5.2.2) are RL-competitive. The first is a novel approach we put forward. The second represents a standard OR strategy, if we leave out the indirect action principle. Though MCTS can be made RL-competitive, no such approach was encountered in RL scheduling literature.

5.2.1 Simulation Search

SimSearch is a simple yet efficient algorithm inspired by the popular RL design using priority rules as actions. In such contexts, RL agents predict the best priority rules for the current state based on their past experience. SimSearch replaces the prediction step with a simulation step.

Concept: The SimSearch scheduling loop depicted in Figure 5.1 and reflected by Algorithm 2 works as follows. The algorithm requires a simulation, a set of n algorithms $\mathcal{A}$, the completion percentage p defining the rollout end, and an optimization target as inputs on initialization. Given the agent-state S_A^t at time t, SimSearch first retrieves a deterministic copy of the simulation in the current state. We use S_E^t to underline the distinction between the environment and agent-state, since the former has a transparent view of future stochasticity while the latter does not. SimSearch then replicates the deterministic copy n times, and rolls out the copies using the different algorithms $\mathcal{A}_i$. For each rollout, the simulation is ran until p of the unprocessed operations in S_A^t are completed. The scores achieved by $\mathcal{A}_i$ are then measured on the n states reached after the rollouts. The algorithm with the best score is selected to choose the action $a_t := \mathcal{A}_{\text{best}}(S_A^t)$ for the current time step. Finally, the chosen action is passed to the original simulation to generate the next state S_A^{t+1}. This outer loop continues until a terminal state is reached by the simulation.

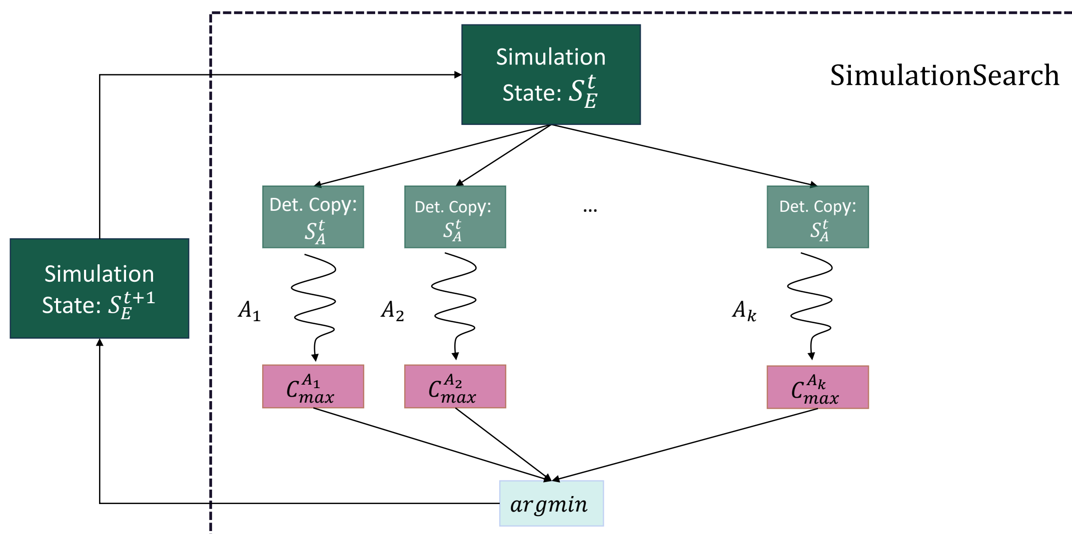


Figure 5.1: Scheduling control using SimSearch for makespan ($C_{\max}$) optimization: Rollouts are performed using algorithms $\mathcal{A}_i$ from agent-states S_A^t containing no stochasticity information. The algorithm with the best score at the end of the rollout is then chosen to select action a_t which is passed to the simulation. The simulation performs the action and produces the next state.

The elements of the set $\mathcal{A}$ can be any scheduling algorithm. To maintain the comparability with RL methods in literature and because for production scheduling simple heuristic solutions can be quite strong, we use priority rules exclusively in our experiments. For simplicity we we did not distinguish between different simulation modes (e.g. job routing and sequencing) in Algorithm 2. This assumes that algorithms $\mathcal{A}_i \in \mathcal{A}$ can deal with the different decision types accommodated by the particular scheduling instance. When using heuristics, this would mean that $\mathcal{A}_i$ is defined by a pair of priority rules, one for job routing the other one for sequencing.

SimSearch's runtime depends on both the simulation and the algorithm set runtimes. The worst case is defined by $O(|\mathcal{A}|(n f_{\mathcal{A}}(S_A^t) f_{\text{sim}}(a_t) + f_{\text{copy}}(S_A^t)))$. Here n denotes the number of operations in the current state, $f_{\mathcal{A}}$ is the runtime of the slowest algorithm in $\mathcal{A}$, $f_{\text{sim}}(a_t)$ is the runtime of the simulation's state-transition function and $f_{\text{copy}}(S_A^t)$ is the overhead

Algorithm 2 Scheduling with SimulationSearch and FabricatioRL

Input: sim, $\mathcal{A}$, p, target
1: state ← sim.reset()
2: done ← $false$
3: **while** ¬ done **do**
4: dc ← sim.get_deterministic_copy() ▷ SimSearch start
5: scores ← []
6: **for** $i \in \{1, \ldots, |\mathcal{A}|\}$ **do**
7: dca ← dc.copy()
8: s_t ← state
9: **while** s_t.completion_lvl < p **do**
10: s_t ← dca.step($\mathcal{A}[i](s_t)$)
11: **end while**
12: scores ← scores + [s_t.target]
13: **end for**
14: i ← argmin(scores)
15: $a_t \leftarrow \mathcal{A}[i]$ ▷ SimSearch end
16: state, _, done, _ ← sim.step(a_t)
17: **end while**

associated with copying the simulation. As noted in the previous chapter, our simulation has, start to finish, a loglinear runtime, $O(n\log(n))$, in the number of operations n. A single simulation step requires logarithmic time. The overhead associated with heuristics choosing an action is linear with respect to the number of scheduling instance operations in the worst case ($O(n)$), as dictated by the time required to find the minimum/maximum in an unsorted sequence. The copy overhead is linear with respect to the state size, which in turn has a constant size proportional to n, hence $f_{\text{copy}}(S_A^t) \in O(n)$. All in all this leads to the following identities:

$$\begin{aligned}\text{SimSearch} &\in O(|\mathcal{A}|(n \cdot f_{\mathcal{A}}(S_A^t) \cdot f_{\text{sim}}(a_t) + f_{\text{copy}}(S_A^t)))\\ &= O(C_1(n^2 \cdot \log(n) + C_2 n))\\ &= O(n^2\log(n)) \end{aligned} \tag{5.12}$$

Furthermore, SimSearch is embarassingly parallel. Individual rollouts could be executed in distinct threads, with no inter-process communication required. This leads to an easy algorithm optimization.

Implementation: The SimSearch implementation is split between the SimulationSearch and SimulationSearchControl classes. The latter implements our control interface and, hence, the play_game method, using the former to select actions via the get_action method. The former implements the Optimizer interface. As such SimSearch could be used as an indirect action. The particularity of this Optimizer is that it must contain a copy of the simulation it is potentially passed to. To break this circular dependency, SimSearch can be initialized without a simulation copy and passed to a new environment as an optimizer. The now instantiated simulation can then be copied and assigned to the corresponsing

SimulationSearch attribute.

Furthermore, simulation-based optimizers need to be updated when another optimizer has made a decision such that the internal simulation reflects the external simulation state. To this end, FabricatioRL's Optimizer interface was extended to contain the additional update method. This method takes a direct action a parameter and does not return anything. Just before the simulation's step function returns, update is called on all embedded optimizers, if any, with the direct action associated with that particular step call. Simulation-based methods use the direct action to synchronise the internal simulation.

5.2.2 Monte Carlo Tree Search

MCTS is a method for approximating optimal decisions in artificial intelligence problems, typically move planning in combinatorial games. It combines the generality of random simulation[2] with the precision of tree search. Research interest in MCTS has risen sharply due to its spectacular success with board games such as Go as a component of AZ.

MCTS' application extends far beyond the world of (adversarial board) games. As long as a problem can be modeled as a game, i.e. a finite sequence of (state, action) pairs with a value associated with at least the last state in the sequence, any problem can be solved using it. As such MCTS is a problem agnostic approach.

As with RL, a simulation accepting an action as an input and returning the next state as determined by the action must be made available to the algorithm. The simulation is used for the purpose of rolling out the game to asses the value of the current state based on the value of an end state reachable from it.

Concept: The "game tree" is the central MCTS data structure. Nodes herein correspond to game states and edges to state-transitions/actions. Generally, from any particular state, more than one action is feasible. If we were to represent states as nodes and recursively expand all nodes according to the actions possible in them until an end state is reached, we would have exhaustively searched the space of all possible move combinations. Having done that, the optimal move could be selected by choosing the action leading down the game tree path associated with the best outcome. This is not tractable for complex planning problems such as Go or, in our case, complex scheduling problems. Instead, a clever way of focusing solely on the most promising paths is needed. Using MCTS to find a solution to a problem corresponds to probabilistically searching the most auspicious portions of the game tree.

The four phases characteristic of the MCTS are the similar to the ones introduced when discussing AZ in Section 4.2.2. The traditional MCTS stages are selection ①, expansion ②, simulation/roll-out ③ and backpropagation/backtracking ④, as depicted in Figure 5.2. Aside from the game state, every node in the tree stores a value and a visit count. The search starting point is a tree consisting solely of the root node of the game, i.e. the initial state. A partial game tree is then constructed as follows:

[2]Monte Carlo algorithms are a category of stochastic algorithms in which the run-time is deterministic whereas the solution quality is a random variable. Higher run-times generally lead to more better solutions.

(1) During *selection*, the current partial tree is traversed by always selecting the child node with the biggest associated score. This score is calculated through a combination of the node value and the number of visits.

(2) Having reached a leaf node, an action possible from the game state associated with it is chosen at random. The node corresponding to the state the action leads to is added as the new leaf node, thereby *expanding* the tree.

(3) During *roll-out*, a simulation with legal actions chosen at random is executed starting at the new node.

(4) When the simulation reaches a terminal state, the end-game result r is used to update the new node, as well as the value of the nodes up the selection path. The visit counts are also incremented by one along the path. The value and visit updates form the *backtracking* step.

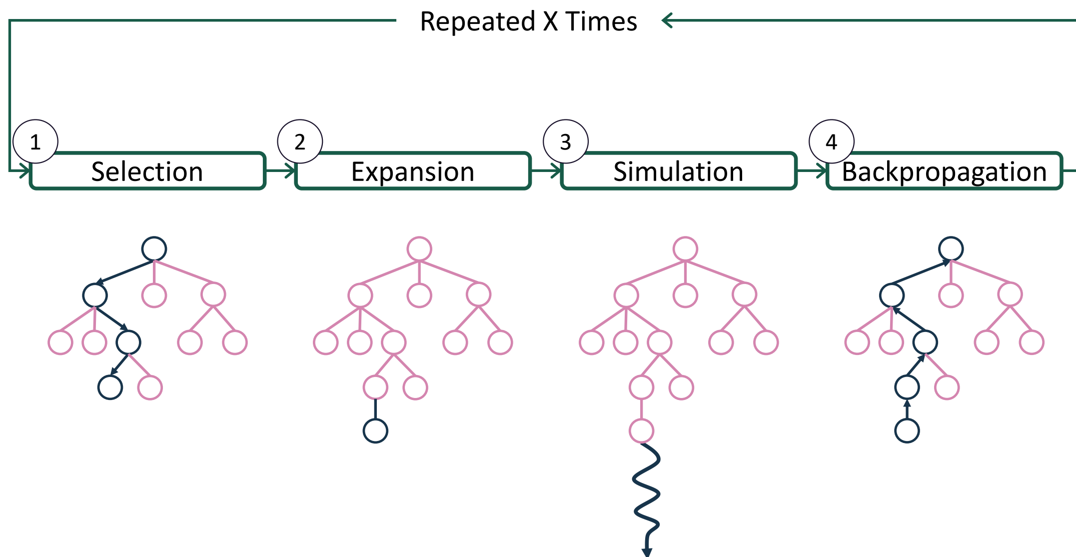


Figure 5.2: The Monte Carlo Search Tree Algorithm. A partial game tree is constructed by repeating selection, expansion, simulation and backpropagation X times. During the Selection phase the current partial tree is traversed from the root to a leaf using values stored in the nodes to decide the path. During expansion, a new node is added to the leaf by choosing a random action and saving the resulting state. Then, a simulation taking random actions is used to reach an endgame state. The value associated with the endgame state is used to update all nodes up the selection path. After X iterations, the best action relative to the root node, is chosen using some selection criteria. (Source: Winands, 2015)

There are are several possible selection strategies (1) for MCTS, such as Bandit Algorithm for Search Trees, Exploration-Exploitation with Exponential Weights or Upper Confidence Bound 1 (UCB1)-tuned (Browne et al., 2012). By far the most popular is the one derived from the UCB1 algorithm introduced by Auer et al. (2002). The UCB1 derived selection strategy is called Upper Confidence Bound for Trees (UCT) (Kocsis et al., 2006), and can be expressed as follows. From a set I of candidate nodes, under the current node p, select best node b using Equation 5.13 (Winands, 2015), where v_i represents node i's value derived from and updated through simulation. v_i should be kept in the Interval $[0, 1]$. C is a

tunable exploration parameter. n_p is the visit count of the current node and n_i the visit count of the ith child.

$$b = \underset{i \in I}{\text{argmax}} \ (v_i + C)\sqrt{\frac{ln(n_p)}{n_i}}) \tag{5.13}$$

Expansion and rollout strategies ② are studied less extensively. While different expansion strategies, such as expanding more than one node at a time, or prohibiting expansion until a certain threshold of simulations have been run through it, exist, their effect on the algorithm's playing strength is rather small (Winands, 2015).

In contrast, during the simulation/play-out step ③, a smart simulation strategy has the potential to significantly improve the algorithm's results (Gelly et al., 2007). The idea behind designing a play-out strategy is to use heuristics to filter out implausible moves. Filtering during play-outs can have two negative effects on the algorithm's performance. On the one hand, over-filtering can negatively impact the MCTS' exploration capability, leading to potentially interesting nodes not being visited. On the other, depending on the computational expensiveness of the heuristic, this can impair the algorithm's speed. As such, designing a simulation guiding strategy appropriate for the task at hand is a delicate matter.

As with the previous steps, different strategies can be used for backtracking ④. These strategies pertain to how the node values v_i are updated up the selection path using the end scores r_t^i. However, generally the simple score average described by Equation 5.14 is used (Coulom, 2006).

$$v_i = \sum_{t \in \{1,\dots,n_i\}} \frac{r_t}{n_i}. \tag{5.14}$$

After the four steps were run a fixed number of times, a move has to be selected starting from the root node of the now partially constructed game tree. The final move selection occurs in accordance with one of four strategies. These are max-child (i), robust-child (ii), robust-max-child (iii) and secure-child (iv):

(i) The first strategy selects the node with the maximum value v_i as the next move.

(ii) Robust-child selects the node with the maximum visit counts.

(iii) The third strategy selects the node that has both the highest visit count and the highest value. If no such node exists, supplementary play-outs are run until such a node is determined.

(iv) Finally, secure child chooses the node that maximizes $v_i + \frac{A}{n_i}$, where A is a parameter, and n_i is the node's visit count

Winands, 2015 note, that "the difference in performance between these strategies is limited when a significant number of simulations for each root node have been played".

MCTS offers a number of advantages over other search strategies. Firstly, though it can be enhanced with domain knowledge during the simulation step, MCTS is domain independent, and can be used as a solver for any accordingly modeled problem (i.e. an iterative process where decisions are made based on a state representation). Secondly, the game tree resulting from MCTS is asymmetric, with more emphasis being lain (more visits) on areas of the tree leading to potentially better results. Lastly, the algorithm is easy to implement and tweak.

There are two major disadvantages to using MCTS in its basic form (no domain knowledge). For one, it can fail to find optimal solutions for medium sized planning problems in a reasonable amount of time. This is a common problem for most search approaches confronted with a vast solution space. While the stochastic principle embedded in the approach somewhat alleviates this issue, it cannot completely eliminate it. The Second problem, namely the algorithm's run-time, is derived from the first. Given a large combinatorial optimization problem instance, a large number of simulations is required to find good solutions, which negatively impacts speed. Though the use of domain knowledge during play-offs can reduce the number of required simulations, the speed-accuracy trade-off cannot be eliminated completely.

Implementation: For our experiments we use FabricatioRL with indirect actions and made the following choices with respect to the MCTS steps. We use the UCB1 meta-heuristic, a full expansion strategy, random rollouts, value averaging during backtracking and an alternative final move selection strategy, we call "max/robust-child". Max/robust-child selects the node with the highest value to visit ratio.

Algorithm 3 presents the MCTS variant implemented for this work. The input is an initial (agent) state S_A^t, an instance of FabricatioRL, a number of iterations x and the exploration parameter C. The tree node i stores a representation of the state S_A^i it corresponds to, the value v_i, the node's visit count n_i, a pointer to the parent node, the action that lead to it and the moves possible from it. Additionally, every node implements the add_child function to link a new child node to itself. The simulation is used for executing moves (step) and for getting the moves possible from a given state (get_legal_actions). Every new iteration the simulation is reset to the initial state S_A^t. The full expansion-policy is forced during the selection loop, where the movement down the tree path is curtailed, if there are still unexplored nodes for any particular state.

The Optimizer-Control dichotomy is in place for our MCTS implementation as well. The MCTS class implements Algorithm 3 within its get_action method. As with Simulation-Search, MCTS objects can dub as indirect actions within FabricatioRL. Here too the MCTS internal simulation needs to be updated to reflect the external simulation state. The updates work as described at the end of the previous section. The MCTSControl class implements the play_game method using the optimizer object to select the action to step through the simulation from start to finish.

Algorithm 3 UCT Monte Carlo Tree Search

Input: S_t, sim, x, C

1: root ← Node(S_t, ⊥, ⊥, sim.get_legal_actions()) ▷ Root; dummy parent and move
2: $i \leftarrow 0$
3: node ← root
4: **while** $i < x$ **do**
5: sim ← sim.get_deterministic_copy()
6: **while** node.child_nodes ≠ ∅ **and** node.untried_moves = ∅ **do** ▷ Selection (1)
7: node ← $\underset{\text{child} \in \text{node.child_nodes}}{\text{argmax}} \frac{\text{child.value}}{\text{child.visits}} + C\sqrt{\frac{\text{node.visits}}{\text{child.visits}}}$
8: sim.step(node.move)
9: **end while**
10: **if** node.untried_moves ≠ ∅ **then** ▷ Expansion (2)
11: move $\leftarrow^{rand}$ node.untried_moves
12: S ← sim.step(move)
13: child ← Node(S, node, move, sim.get_legal_actions())
14: node.add_child(child)
15: node ← child
16: **end if**
17: **while** sim.get_legal_actions() **do** ▷ Roll Out (3)
18: moves ← sim.get_legal_actions()
19: move $\leftarrow^{rand}$ moves
20: S ← sim.step(move)
21: **end while**
22: v ← −sim.core.state.system_time
23: **while** node ≠ root **do** ▷ Backtracking (4)
24: node.value ← node.value $+v$
25: node.visits ← node.visits +1
26: node ← node.parent
27: **end while**
28: $i \leftarrow i + 1$
29: **end while**
30: contenders ← node.child_nodes
31: best_node ← $\underset{\text{child} \in \text{contenders}}{\text{argmax}} \frac{\text{child.value}}{\text{child.visits}}$ ▷ Final selection
32: **return** best_node.move

5.3 Remarks on Baseline RL-Competitiveness

Strictly speaking, the two snapshot-based approaches we introduced are not fully RL-competitive. Priority rules, while adaptive, efficient in terms of runtime and requiring no mathematical modeling overhead do not always yield good quality solutions in the deterministic case. Conversely, the constraint programming heuristic we created, CP3, yields better solutions, is adaptive and runtime efficient, but requires mathematical modeling.

Bear in mind, that the mathematical modeling overhead associated with CP is partially amortized by the subsumption relationship characterizing scheduling setups. Creating a CP model for a general setup yields a solver for all the subsumed setups. This property is akin to RL transferability.

Simulation-based methods do have a caveat: To reach a decision, they not only require knowledge about the current production state, but, as the name suggests, also a full simulation of the production process. This however, is a caveat of RL as well. While model-free RL approaches do not require simulation during deployment, the training process cannot be decoupled from simulation.

CHAPTER 6

Choosing through Experimentation: Setups, RL Design, Training and Evaluation

No amount of experimentation can ever prove me right; a single experiment can prove me wrong.

— Albert Einstein

This section is focused on the experimental evaluation of RL algorithms for production scheduling and serves to bind all previous sections together. Chapter 2 informs our entire experiment design from scheduling setup choice, through RL employed model design, to our chosen validation scheme. We use the simulation framework described in Chapter 3 to make sure that our experiment setups are reproducible, such that other authors can easily test different approaches against the results presented here. We evaluate our chosen RL algorithms, DDQN and AZ, which were described in Chapter 4, by pitting them against the baselines constructed in Chapter 5.

Our experiments are tailored to address the *stochastic experiments*, *stochastic benchmarking setup* and *RL design* gaps from Section 2.5. We first transparently construct two stochastic production scheduling setups in Section 6.1. In particular we assess the system behavior and argue for our parameter choices such that the differences between the resulting benchmarks become clearer. In Section 6.2 we present several design choices framing our chosen RL algorithms. Here, we also describe two novel approaches to experimentally evaluate rewards and scheduling state features without requiring model training. Finally, we run our model selection experiments and evaluate the results of the best models in Section 6.3. The model selection experiments have the dual purpose of evaluating both design and RL algorithm parameter choices.

Our experiments are designed in such a fashion that the adaptivity and transferability (see Table 4.1) of both RL algorithms and our baselines are tested, thereby assessing the RL algorithm advantages. Highly adaptive algorithms are expected to perform well in

stochastic environments, such as those constructed in this section. The transferability property is evaluated by testing both RL and the RL-competitive baselines on different scheduling problems. Our RL algorithms are trained on one type of problem only.

6.1 Scheduling Setup

19 of the 98 papers reviewed in Chapter 2 applied RL solutions to the flexible job-shop scheduling problem (FJc) making this problem the second most popular production setting considered, just after the "classic" job-shop scheduling setup (Jm). In terms of additional constraints, M_i^o and r_j^s are the most popular, with 21 and 24 papers respectively considering them. In terms of optimization goals, 56 of the publications are using makespan $C_{\max}$ as their target, making the latter the most popular optimization goal. This popularity analysis lead us to choose the setups $(FJc|r_j^s, M_i^o|C_{\max})$ and $(Jm|r_j^s|C_{\max})$ as our algorithm battlegrounds.

In this section, we construct the arena where RL is to compete against simple priority rules, our CP heuristic, SimSearch and MCTS. First, we briefly introduce the deterministic base scheduling problems $(Jm||C_{\max})$ and $(FJc||C_{\max})$ and the structure of the available benchmark instances in Section 6.1.1. We then proceed to create the dynamic version of the problems by expanding the base setups in Section 6.1.2, where we also investigate the resulting dynamic system behavior in terms of workload.

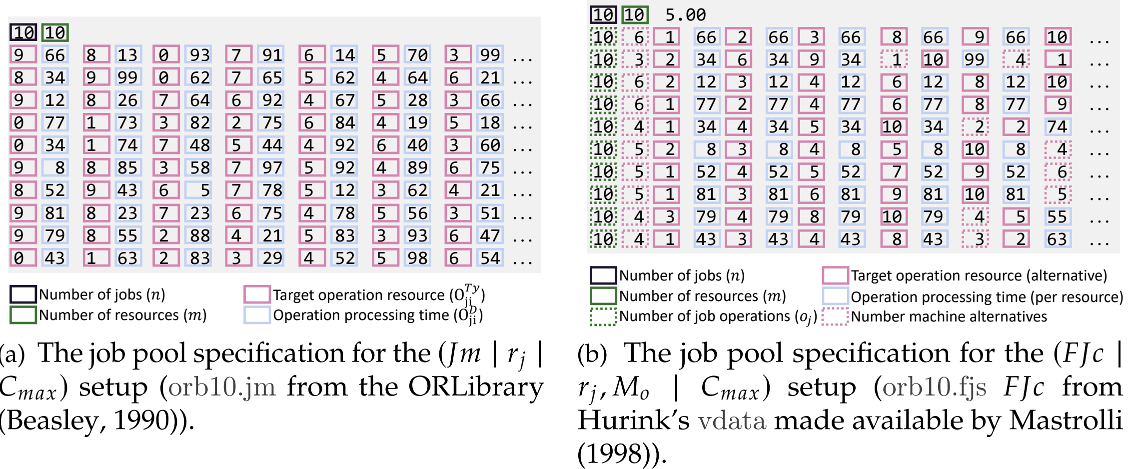
6.1.1 Base Scheduling Problems

There are many available benchmarking instances for both $(Jm|)$ and $(FJc|)$ setups. Note however that, for FJc the separation between $(FJc||C_{\max})$, $(FJc|M_i^o|C_{\max})$ and $(Rm + FJc|M_i^o|C_{\max})$ is not made clear in the benchmark datasets. Also, note that there is also a correspondence between the job structure of some Jm and FJc instances. This is intuitive since the latter can be created by simply expanding single machines into work centers or expanding the operation capabilities of individual machines without changing the job structure.

We selected orb10.jm from the ORLibrary (Beasley, 1990) and the corresponding orb10.fjs from the repository maintained by Mastrolli (1998) as the $(Jm||C_{\max})$ and $(FJc|M_i^o|C_{\max})$ to extend. The selection of the Jm instance was done randomly from the set of instances with 10 machines and 10 jobs in the OR Library. We then looked for the FJc instances (Mastrolli, 1998) which were built using the orb10.jm job structure and found three such instances with an average of 1.1, 2 and 5 machine alternatives per operation respectively (Hurink's edata, rdata and vdata). We chose the one with the most alternatives (i.e. vdata).

Figure 6.1 shows the instance representations for Jm and FJc respectively. For Jms the representation is fairly straightforward (Figure 6.1a). The first row contains the white space separated number of jobs n and the number of machines m. Every subsequent row represents a job and contains the n operation type and operation duration pairs.

FJc instances (Figure 6.1b) are constructed similarly. Additionally, to n and m, the first row



(a) The job pool specification for the $(Jm \mid r_j \mid C_{max})$ setup (orb10.jm from the ORLibrary (Beasley, 1990)).

(b) The job pool specification for the $(FJc \mid r_j, M_o \mid C_{max})$ setup (orb10.fjs FJc from Hurink's vdata made available by Mastrolli (1998)).

Figure 6.1: Base Jm and FJc instances at a glance.

contains the average number of machine alternatives. The subsequent rows represent jobs and adhere to the following specification. The first row entry represents the total number of operations o_j within the job. The row is continued by o_j variable length operation specifications. An operation specification contains the number of alternative machines m_{ji}^{alt} at the first position. This information is followed by m_{ji}^{alt} number pairs which indicate the resource index and operation duration for each alternative resource.

The different operation types can be inferred from the number of distinct resource alternative sets present in the specification. The type of scheduling problem can then be identified by looking at whether or not these resource sets overlap. The setup at hand corresponds to a $(FJc \mid M_o)$, since there are 97 distinct alternative sets with plenty of overlap.

6.1.2 Dynamic Scheduling Problems

The dynamic problems $(Jm|r_j|C_{\max})$ and $(FJc|r_j, M_i^o|C_{\max})$, which represent our algorithm test-bed are constructed based on the static problems described above. Both our problems consist of a stream of 100 jobs, with the first 14 jobs being known at time 0 and the following 86 being revealed as time progresses. The number of jobs was set using the reviewed literature for orientation. To create the new arrival stream, we sample job specifications uniformly at random with replacement from the pool of jobs defined by the static scheduling problem instances described above. To better understand the resulting scheduling problems, we need to discuss the WIP parameter as well as the choice and effects of the job release times r_j.

Number of Jobs: The 100 total jobs were derived based on problem sizes in literature. Figure 6.2 shows the distribution of the total number of operations for 62 of the 98 papers reviewed in Chapter 2. We used the size information in Table A.1 to compile the plot data. In a preliminary step, we eliminated 33 papers containing inclarity in terms of the problem size and further three representing obvious outliers (more than 10000 operations considered). From the figure we can see that the average number of operations is close

to 1000 which also corresponds to the upper quartile of the data. Since our jobs contain 10 operations each, we limited the number of jobs to 100 to approximately match the larger instance sizes in literature. Note that the problem size is no direct indication of the problem's combinatorial complexity.

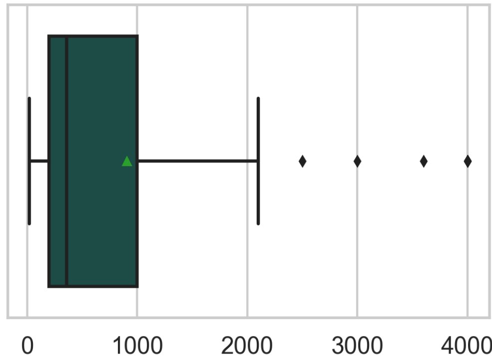
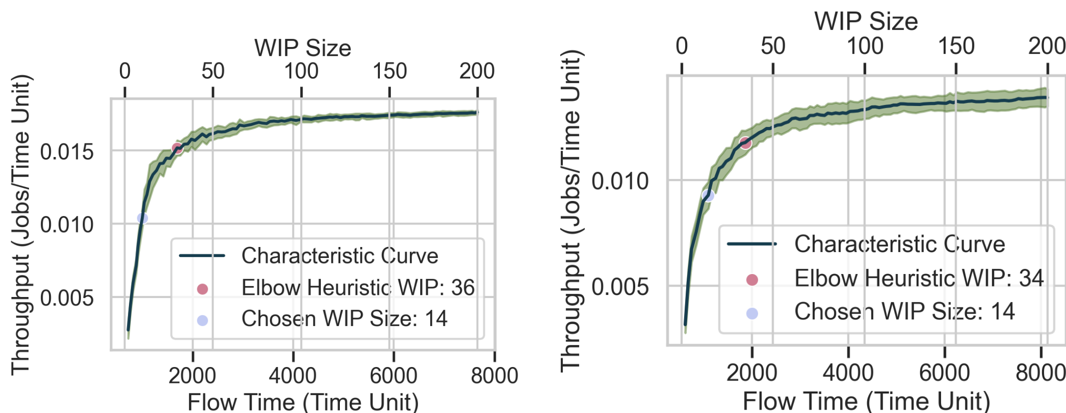


Figure 6.2: Distribution of the problem size in terms of operation numbers in RL scheduling literature.

WIP: In online scheduling problems, a WIP is often used to control the trade-off between flow time and utilization/throughput. The WIP limits the number of jobs seen by the scheduling system to a fixed number. Jobs outside the WIP, e.g. new job arrivals, have to wait for a WIP slot to open up before being added to the window. WIP slots become available on completion of WIP jobs. Flow time $F_j := C_j - s_{j1}$ is the time between the processing start time s_{j1} of the first operation from job j and the job completion time C_j (cmp. Section 2.2.3). A large WIP leads to an increase in machine utilization, and consequently throughput, at the expense of an increasing flow time (Choo, 2017). Conversely, small WIP lead to shorter flow times at the expense of utilization. There is no standard way of choosing the WIP size.

The WIP impact on production KPI is setup and control dependent and subject to research. While studies of the WIP impact on particular production setups exist (e.g. for Fm Yang et al., 2006; Lee et al., 2016) to the best of our knowledge, no setup and control independent formulation is as of yet known. For Fm, the typical WIP selection procedure involves plotting average flow time against average throughput, the so-called system characteristic curve, and selecting the WIP at the point of maximum curvature (Yan et al., 2019). Figure 6.3 visualizes this approach.



(a) Trade-off between utilization and flow time for the FJc setup.

(b) Trade-off between utilization and flow time for the Jm setup.

Figure 6.3: WIP size choice. The WIP was selected at the elbow of the Flow time-utililzation tradeoff curve.

For the problems at hand, we use FabricatioRL to choose the WIP size. First, we defined 200 different Jm and FJc instance sizes by varying the number of jobs n from 1 to 200. For each n we created 20 different instances by sampling the appropriate number of jobs uniformly at random from the pool of 10 jobs defined by orb10.jm and orb10.fjs respectively. Subsequently, we ran the simulation with a simple sequencing heuristics, namely SPT, and a simple job routing heuristic namely LQT as a control on all Jm/FJc instances generated. FabricatioRL was parameterized with a WIP size equal to n, meaning that all jobs visible to the control system for all instances. We then noted down the makespan $C_{\max}$ and average flow time F_{ave} achieved by the heuristic controls on each of the 4000 instances. We grouped these values by the number of jobs in the instance. By averaging the values inside the groups we obtained 200 $C_{\max}^{\text{ave}}$-$F_{\text{ave}}^{\text{ave}}$ pairs.

We sorted the $C_{\max}^{\text{ave}}$-$F_{\text{ave}}^{\text{ave}}$ pairs by the number of group jobs increasingly and plotted the average flow time against the average throughput, i.e. $\frac{n}{C_{\max}^{\text{ave}}}$, as per Yan et al. (2019) (Figure 6.3). We used the "Kneedle" algorithm (Satopaa et al., 2011) to find the number of jobs corresponding to the point of maximum curvature. This yielded the values of 36 and 34 jobs for the FJc and Jm systems respectively. These points would correspond to prioritizing throughput and flow time equally in the system. However, we decided to prioritize flow times over throughput. As such we chose the value of 14 jobs as the WIP window size for both problems at hand.

Release Times: The definition of job-arrival times is essential with respect to scheduling system behavior. Depending on how these times are chosen, the system can be under- or over-booked, which can significantly impact the scheduling problem and hence the performance of different solution approaches.

However, details with respect to release times definition are mostly scarce in the related scheduling literature. Wang et al. (2007) sample release times from an exponential distribution with a mean of 3 and 4 for "light" and "heavy load conditions" respectively. The authors do not elaborate on what constitutes a "heavy load condition". Similarly, Luo et al. (2021b) and Luo et al. (2021a) use a Poisson distribution for arrival times selecting the distribution parameter λ uniformly at random from $\{25, \dots, 100\}$ and $\{50, \dots, 200\}$ respectively without detailing the choice of the distribution or its effect on the system. Waschneck et al. (2018) also do not detail the release time distribution nor its impact on the system behavior at all.

Central to defining job arrival times r_j^s are the times required for one WIP slot to get freed r_j^{slot}, which are dependent on the WIP job structure and scheduling mechanism. One could experimentally determine r_j^{slot} through simulation with a heuristic control on a sufficient number of randomly sampled scheduling instances of n jobs, where n corresponds to the WIP size. Job arrival times could then be sampled from the resulting empirical distribution of r_j^{slot}.

To defining r_j^s we construct the following heuristic requiring a single simulation run on an instance with a number of jobs equal to the WIP size. Our heuristic requires saving the resource utilization Utl_t and the current WIP fill level with respect to operations w_t^{ofl} at

every decision time point t during the simulation run. At the end of the run, we calculate an empiric r_j^{slot} approximation using flow times F_j, a utilization aggregate Utl_{agg}, $w_{\text{ave}}^{\text{ofl}}$ and the number of machines m (Equation 6.1). We then use r_j^{slot} to parameterize a truncated normal distribution from which we sample inter-arrival times r_j^{ia} (Equation 6.2). Finally, we set the arrival times for jobs j as the cumulative sum of the first j inter-arrival times (Equation 6.3).

$$r_j^{\text{slot}} \approx F_j \cdot \frac{w_{\text{ave}}^{\text{ofl}}}{Utl_{\text{agg}}} \cdot \frac{1}{m} \cdot 0.9 \tag{6.1}$$

$$r_j^{\text{ia}} \sim N(\mu(r_j^{\text{slot}}), \sigma(r_j^{\text{slot}}), r_{\text{min}}^{\text{slot}}, r_{\text{max}}^{\text{slot}}) \tag{6.2}$$

$$r_j^{s} := \sum_{i=1}^{j} r_i^{\text{ia}} \tag{6.3}$$

The heuristic is based on the intuition that flow times are directly proportional to r_j^{slot}. Using F_j alone to estimate r_j^{slot} is insufficient since r_j^{slot} is additionally affected by the machine utilization Utl_i and $w_{\text{ave}}^{\text{ofl}}$. Utilization is inversely proportional to r_j^{slot}. If the utilization is high, the slot release times decrease which is why we use Utl_i to scale F_j. $w_{\text{ave}}^{\text{ofl}}$ serves to adjust our formula to the workload present in WIP. Since a higher workload correlates with a higher slot release time, we use this value as a multiplicative factor. Finally, m accounts for processing parallelization on different resources and 0.9 is an experimentally determined adjustment factor.

We can partly control whether the system is overbooked, under-booked, or more or less balanced by using the max, min, or ave operator as the aggregation function in Equation 6.1 above. We parameterized the FJc and Jm setups with the $mean$ and max aggregates respectively.

System Behaviour: To keep with the frame of this work, we only inspect the system's behavior with respect to aspects of system load, namely the WIP fill level, and the machine utilization. Both aspects are directly dependent on the WIP size and release times parameters. The WIP fill level, i.e. the ratio between the total number of jobs in the system and the WIP size, takes values greater than one if the system is overbooked and smaller than one if under-booked.

To investigate the effects of our choices, we created 1600 problem instances for each of the two scheduling setups using the simulation described in Chapter 3. The simulation was then run on these inputs using SimSearch as a control. In each state, we store the WIP fill level, resource utilization, and system time along with 47 further state features (see Section 6.2.3). These values are conveniently tracked by our simulation. Note that we use the data acquired in this fashion in the following sections as well. Figure 6.4 shows the full data acquisition loop.

Figures 6.5 and 6.6 display the effects of our parameter choices (WIP size and r_j^s) on the

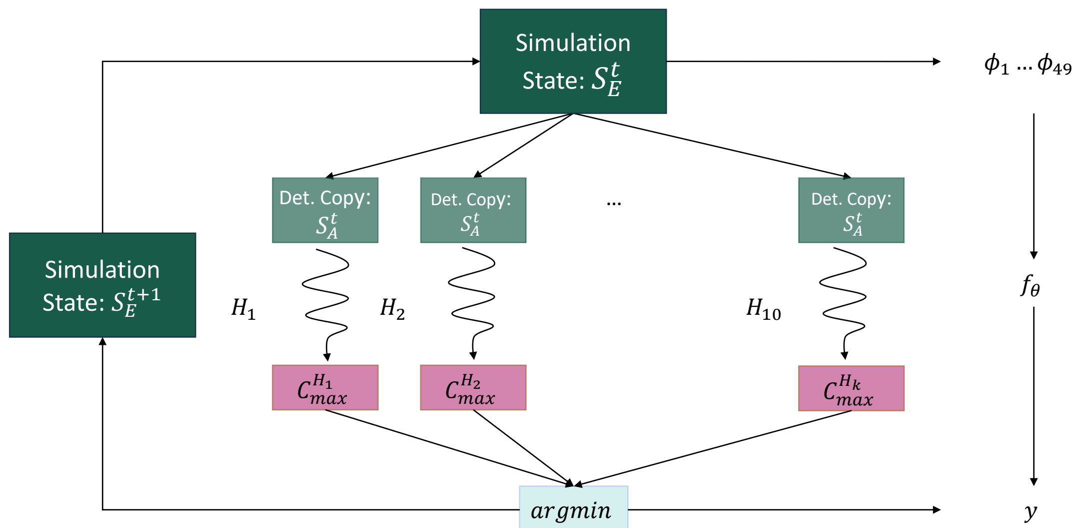
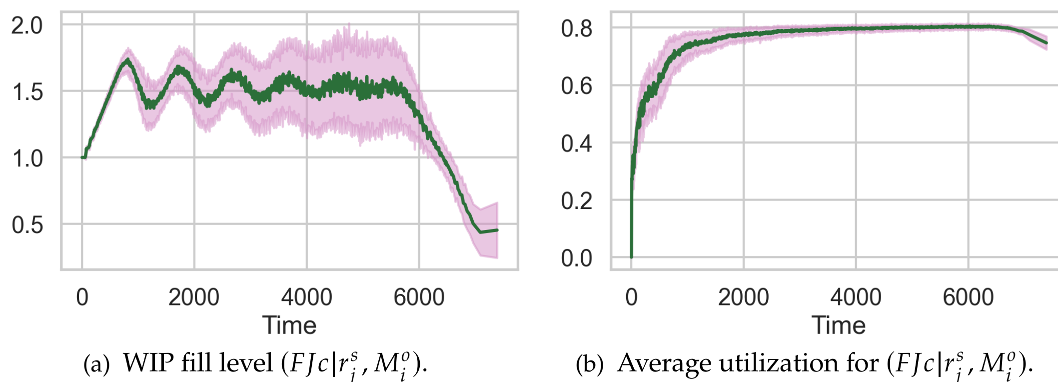


Figure 6.4: Data acquisition loop using simulation search. For all states, features $\phi_1, \dots, \phi_{49}$ are first extracted. Then simulation search is being ran to select the best optimizer for the current state. In this case the optimizers correspond to the ten sequencing heuristics from Section 5.1.1 combined with the LQT job routing heuristic. the winning heuristic y is saved and associated with the features. Models f_θ can be used to predict the y.

$(FJc|, r_j^s, M_i^o)$ and $(Jm|\ r_j^s)$ setups respectively. The green line in all plots indicates the average y-axis values μ_t for the corresponding point in time t, averaged over the 1600 analysis instances. The plot area around the mean is filled between $\mu_t - \sigma_t$ and $\mu_t + \sigma_t$, where σ_t is the standard deviation of the 1600 y-values for the time point t.

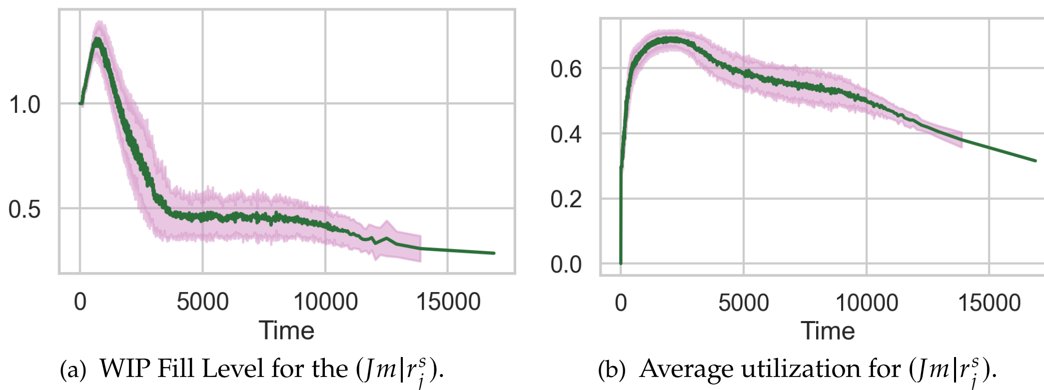


(a) WIP fill level $(FJc|r_j^s, M_i^o)$.

(b) Average utilization for $(FJc|r_j^s, M_i^o)$.

Figure 6.5: System load behavior for $(FJc|r_j^s, M_i^o)$. The system is overbooked and maximizing utilization.

From the fill level plot (Figure 6.5a), which depicts the evolution of the WIP fill level (y-axis) over time, we see that the FJc system is overbooked with the number of total jobs in the system oscillating around 1.5 times the WIP size. The variance of these values between instances gradually increases before decreasing again towards the end of the scheduling game. The system utilization (Figure 6.5b) rapidly increases during the beginning of the scheduling time horizon ($t \in [0, 2000]$) and peaks towards the end of the simulation ($t > 6000$) with mean values slightly over 80%. The utilization increase during the middle part of the simulation is only slight.



(a) WIP Fill Level for the $(Jm|r_j^s)$.

(b) Average utilization for $(Jm|r_j^s)$.

Figure 6.6: System Load Behavior for the $(Jm|r_j^s)$. The curves show clear signs of the system being underbooked.

In Figure 6.6a we see the WIP fill level curve of a mostly under-booked Jm system. The plot shows a steady drop in fill level after an initial peak of 1.5 at around $t \approx 2000$ time units. The drop continues until it reaches a steady value of approximately 0.5 during the middle part of the scheduling horizon ($t \in [4000, 10000]$). There is a further fill level decrease towards the tail end of the scheduling horizon when no new jobs arrive in the system. The system utilization in Figure 6.5b confirms the under-booked system hypothesis with average utilization values decreasing monotonously after a peak of around 65% at $t = 2000$. The figures also suggest the presence of an outlier in the analysis data, since the variance interval ends at $t \approx 14000$. This means that an instance from the 1600 considered has a significantly longer last decision point, namely at $t \approx 16000$.

We make three observations based on the described figures. Firstly, we can see that our inter-arrival time generation heuristic is imperfect, leading to an under-booked system in the Jm case contrary to our expectations. This is in part due to the fact that the heuristic was tailored during experimentation with the FJc system. Nevertheless, in both cases, the job stream is steady during the middle part of the simulation time, with a WIP fill-level oscillating around 1.5 in the FJc case and just under 0.5 in the Jm case respectively. Secondly, while it is hard to infer the maximum utilization time for the Jm setting given the under-booked situation, the utilization plots suggest that the FJc setup can achieve higher utilization percentages, which points to the advantages of the added system flexibility (i.e. resource alternatives) in the FJc case. Lastly, given the radically different system behavior, we can expect control algorithm ranking to differ significantly between systems. In particular, it is improbable that transfer learning would work well between these problems.

6.2 RL Design

In this section we motivate and detail the RL design choices which we consider in our model selection phase (Section 6.3.2). There are many possible MDP breakdown-state-action-reward-agent deployment combinations as was revealed in Chapter 2. Because of the reproducibility and evaluation gaps in RL production scheduling literature, many

of these combinations are in need of (re-)evaluation. However, completing this task is far beyond the scope of this work. To cover as much ground as possible, we select Iterative Sequencing as our MDP breakdown and focus solely on a single agent deployment scheme. The breakdown choice is informed by the overwhelming popularity the Iterative Sequencing breakdown in the RL production scheduling community (see Figure 2.7). The same holds for the single-agent deployment scheme (see Figure 2.13).

We start by laying down our broad state-action design choices and introducing an ex-ante analysis scheme for rewards and state features in Section 6.2.1. We then proceed to reveal the results of the ex-ante evaluation with respect to rewards in Section 6.2.2. In Section 6.2.3 we elaborate on the exact composition of our state spaces and reveal the results of the ex-ante evaluation of state features. Section 6.2.4 is reserved for a discussion of our action design.

6.2.1 Design Space Reduction

Focusing solely on the Iterative Sequencing breakdown significantly reduces our design space. However, FJc instances require job routing decisions alongside sequencing, since operations can be executed on one of several machines. Instead of taking both job routing and sequencing decisions with RL (Interlaced Routing and Sequencing) we use LQT for all job routing decisions. This approach is complementary to the approach taken by Kuhnle et al. (2020), where the authors fixed FIFO as a sequencing heuristic and only took job routing actions using RL agents.

In terms of state space-action space combinations we take the following three approaches into consideration:

- D1 Firstly we offer a part of the *raw state* information to agents and have them take *direct actions* consisting of *operation indices*.
- D2 Secondly, we feed the same *raw state* information into our agents but model their action-space such that an action corresponds to a *job index*.
- D3 Lastly, we construct and use global state features for the state-space based on which agents need to select a sequencing heuristic from a fixed set.

These three designs cover both the raw and feature state categories and both direct and indirect action paradigms. Thus, our model selection phase can uncover the challenges associated with all the widespread state-action modeling paradigms within the chosen MDP breakdown. We do not consider a design for the last possible combination, namely feature state-raw actions, since global features obfuscate too much of the scheduling problem's structure for the agent to be realistically expected to learn to pick a correct operation or job index. Also note that a combination of both raw states and features should also be considered in the future.

Using a novel yet simple data-driven ex-ante analysis, we can both *isolate a single reward* to be used for all the above designs and *select a state feature set* for the D3 design without resorting to costly RL model training end evaluation. To select the reward, we gather

data by using heuristics as simulation controls and investigate the correlation of different cumulative rewards with the makespan which represents our optimization goal. Similarly, we select the feature set by using a Random Forest model, which is trained to predict the instance makespan given a particular state, as a feature ranker, thereby isolating the ten most informative features as the D3 input.

The data acquisition loop used for the ex-ante analysis of rewards and state features was introduced in 6.1.2. In addition to the WIP fill level, resource utilization, and the system time information stored for each state during the setup analysis, we now store the raw state information, chosen actions, the "winning" heuristic, the state features, and cumulative rewards. Note that the time between consecutive states varies considerably. To avoid high correlations between states, which can be a problem for agent learning, the data accumulation loop skips recording trivial sequencing decisions, i.e. those where a single operation is present in the resource buffer.

6.2.2 Reward Design

During the data acquisition loop, we record the following eight rewards, $R1$ to $R8$, which we formally define in the equations below. We use $U_i(S_t)$ to denote the utilization time of machine i as recorded by the environment-state s_t at time point t. $U_i(S_t)$ is calculated as the total of the machine processing time divided by the current state time. Similarly, the function $Q_i(S_t)$ extracts the resource buffer length for the resource i from S_t. Allowing a slight abuse of notation, the function $t(S_t)$ extracts the time recorded by the environment-state S_t at distinct time points. Given any variables X_i, $\bar{X}$ represents their mean over the index set. We use the σ operator as a shorthand for the standard deviation over index variables i to avoid the lengthy formulation $\frac{1}{n}\sum_{i=1}^{n} X_i - \bar{X}$. Last but not least, $\mathcal{M}$ represents the set of all resources, $\mathcal{J}$ the set of all jobs, and D the duration of operations currently visible in the WIP window.

$$R1(S_t) := U_{ave}(S_t) - U_{ave}(S_{t-1}) \tag{6.4}$$

$$R2(S_t) := \begin{cases} 1, \text{ if } U_{ave}(S_t) > U_{ave}(S_{t-1}) \\ 0, \text{ if } U_{ave}(S_t) > 0.95 \cdot U_{ave}(S_{t-1}) \\ -1, \text{ otherwise} \end{cases} \tag{6.5}$$

$$R3(S_t) := e^{\frac{U_{ave}(S_t)}{1.5}} - 1 \tag{6.6}$$

$$R4(S_t) := \frac{U_{ave}(S_t) \cdot (t(S_t) - t(S_{t-1}))}{t(S_t)} \tag{6.7}$$

$$R5(S_t) := \begin{cases} 1, \text{ if } \sigma(\frac{U_i(S_t)\cdot(t(S_t)-t(S_{t-1}))}{t(S_t)}) > \\ \qquad \sigma(\frac{U_i(S_{t-1})\cdot(t(S_{t-1})-t(S_{t-2}))}{t(S_{t-1})}) \\ -1, \text{ otherwise} \end{cases} \tag{6.8}$$

$$R6(S_t) := -(t(S_t) - t(S_{t-1}) \tag{6.9}$$

$$R7(S_t) := 1 - \frac{(t(S_t) - t(S_{t-1}))}{\sum_{j,i \in \mathcal{M}\times\mathcal{J}} D_{ji}} \tag{6.10}$$

$$R8(S_t) := -\sum_{i\in\{1,\dots,|\mathcal{M}|\}} Q_i(S_t) \tag{6.11}$$

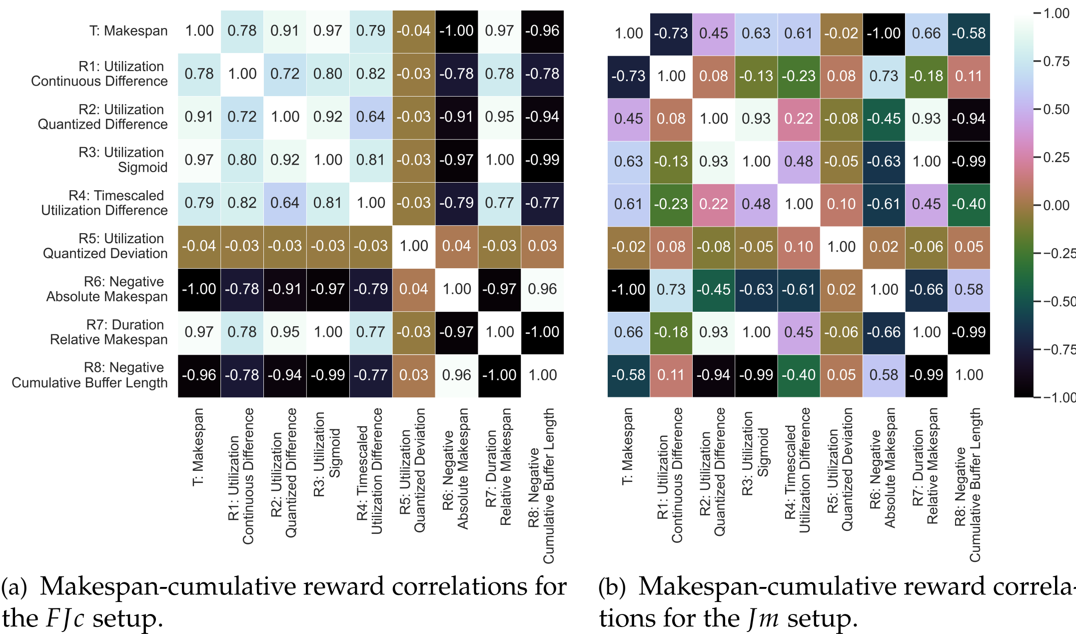
The set of recorded cumulative rewards contains four signals defined in the surveyed literature as well as three new signals defined by us. $R1$ (Equation 6.4) denotes the "Utilization Difference" between two consecutive states. The idea herein is that the agent should strive to increase the utilization as much as possible thereby decreasing the makespan. The "Quantized Utilization Difference" $R2$ (Equation 6.5), introduced by Luo (2020), follows the same principle, with the distinction that the reward between two states is, this time, discrete. The last utilization-based reward from literature is the "Utilization Sigmoid" $R3$ (Equation 6.6), wherein the utilization is passed through a sigmoid function (Kuhnle et al., 2020). Since the time difference between two consecutive states is not constant, we added $R4$, i.e. the "Time Scaled Utilization Difference" (Equation 6.7) to our reward set. The "Quantized Utilization Deviation" $R5$ (Equation 6.8) is yet another signal introduced by us. It punishes high utilization standard deviations and rewards small ones. $R6$ is proposed by Gabel et al. (2012) and represents the negative state-transition time (Equation 6.9). The cumulative version of this reward is equal to the negative makespan, which is why we refer to it as the "Negative Absolute Makespan" . $R7$, which we call "Duration Relative Makespan" represents the negative transition time scaled by the duration of the operations in WIP at the previous step (Equation 6.10). $R8$, which was introduced by Gabel et al. (2007b), returns the "Negative Cumulative Buffer Lengths" at the current point in time (Equation 6.11).

The correlations between the different reward signals and the goal variable depicted in Figure 6.7 underline the difficulty of reward engineering for production scheduling problems: Contrary to our expectation of a strong negative correlation between the cumulative rewards and the makespan optimization goal, in most cases, the reward signals do not correlate or even correlate positively with the target. Additionally, given different setups, the correlation behavior differs. For the FJc setup, the viable reward signals are the negative absolute makespan, $R6$, and the cumulative buffer length, $R7$. In the Jm case, the best contenders are the continuous utilization difference, $R1$, and the negative absolute makespan, $R6$. Since in the D3 experiments we intend to test the learning transferability capacity of our RL models, we select $R6$, which, by design, displays a perfect correlation with the target variable in both the FJc and the Jm cases, as the agent reward.

6.2.3 State Design

With the advent of deep learning, the "raw" versus "features" debate seems to now favor the former, at least in the field of computer vision. This is because CNN can extract meaningful information from raw images offering a convenient alternative to hand-crafted features. This saves researchers the need for domain expertise and development time, offsetting the overhead to the model training time.

In RL scheduling this debate is, as of yet, still open. There are three aspects to be considered when making a design choice in favor of one or the other:



(a) Makespan-cumulative reward correlations for the FJc setup.

(b) Makespan-cumulative reward correlations for the Jm setup.

Figure 6.7: Correlations between different cumulative rewards signals and the makespan goal for the FJc and Jm scheduling setups. Since agents seek to maximize their rewards, and the optimization goal is to lower the makespan, a strong negative correlation is an indicator of reward viability. The correlation behavior of the cumulative rewards varies greatly between scheduling setups.

> First, there is the issue of the overhead associated with developing features. Constructing features is not easy, since both domain expertise and implementation time are required for it. This makes the raw state information design choice more attractive.
>
> Secondly, there is the issue of scheduling setup or at least scheduling problem size invariant models. Features can encode information in a scheduling problem size and and setup independent fashion, making the same models usable in a vaster array of applications, which makes features an attractive design choice.
>
> Thirdly, there is the issue of model training. On the one hand, raw states are generally much larger than feature states which can slow down or even impede learning. On the other hand, during feature construction, essential state information may be missed, which can lead to poorer results.

Raw States D1-D2: The complete raw information set that could be used as the agent-state for the FJc and Jm setups is given by the tuple $(O^P, O^{Ty}, O^D, L, S, M^{ty}, t, c)$, i.e. the operation precedence graph, the operation type matrix, the operation duration matrix, the machine capability matrix, the system time, and the current resource requiring a decision (see Figure 3.7). This would amount to an input size of $n \cdot o^2 + n \cdot o \cdot 4 + m \cdot ty + 2 = 14 \cdot 10^2 + 14 \cdot 10 \cdot 4 + 10 \cdot 97 + 2 = 2932$ for a WIP size of 14 and 97 distinct machine capabilities.

To reduce the input space, we eliminate O^P and M^{Ty} from the agent-state-space. We can safely eliminate O^P since the precedence constraints are all the same for all jobs, and, as

such, constant for both the FJc and the Jm setups. As for the Machine capability matrix, we theorize, that agents could learn the appropriate correspondence between types and machines from data, given enough training iterations. This leaves us with a significantly reduced state size of 562.

Both the D1 and the D2 designs use the vector resulting from flattening and concatenating the elements of the $(O^{Ty}, O^{D}, L, S, t, c)$ sextuple. To flatten the contained matrices, we simply concatenate the contained rows. We use a flat vector rather than the matrices in their original shape because of the fully connected network architecture of our DQN models. While CNN have been used in some publications (e.g. Liu et al., 2020) within our modeling, there are no spatial correlations that CNN can benefit from. Additionally, fully connected architectures were deployed by most authors in the related literature.

Feature State D3: The feature inputs for the D3 experiments were collected in three steps. We first computed 49 features thereby significantly extending the feature sets encountered in related publications. In the second step, we used a Random Forest model trained to predict the makespan target to assess the feature importance for both the FJc and the Jm setup. We selected the ten best features from the FJc setup as inputs for our RL models.

Table 6.1 gives an overview of the features we employed during our experiments. For every feature, we noted down the feature number, its name, its formal definition, and the feature rank as assessed by the feature_importance property of the Random Forest model for the FJc and Jm setups.

The overview table additionally groups the features into four general categories, namely resource-centric, job-centric, structural, and behavioral. The first two categories are mostly literature-inspired (1) while the former two categories are a contribution of ours (2):

(1) Most of the *resource- and job-centric* features we use are inspired by literature, albeit normalized in different fashions. The features pertaining to the buffered operations, which are sometimes referred as “local features”, i.e. $F1$, $F2$, $F3$ and $F4$ are derived from the works of Riedmiller et al. (1999), Aydin et al. (2000), Wang et al. (2005), Gabel et al. (2007a), Chen et al. (2010), Thomas et al. (2018), Zhou et al. (2020), Wang (2020), and Kuhnle et al. (2020). These are by far the most popular indicators employed in RL scheduling approaches. Different flavors of the tardiness features $F8$ and $F9$ were used by Wang et al. (2005) and Luo (2020). The utilization indicators $F5$, $F6$ and $F7$ were also used by Thomas et al. (2018), Park et al. (2019), and Wang (2020). The flow-time-related indicators are based on the works of Luo (2020), Hofmann et al. (2020), and Gabel (2009) The throughput-related features stem mainly from the work of Luo (2020). Finally, our makespan estimate is inspired by Gabel (2009). Note that implementing all the features found in the literature and performing an in-depth analysis thereof is beyond the goal of the current elaboration. As such, it is possible that some of the features in the existing literature were not implemented. Furthermore, there are also some features, e.g. the past number of tool changes or the sum of tooling times at the current machine (Park et al., 2019), the duration of

machine failures (Zhao et al., 2019) or the utilization of vehicles (Kuhnle et al., 2020) that do not apply to our chosen setups.

(2) The *structural and behavioral* categories are developed by us with none of the features therein having been encountered in the related literature. Structural features encode information from the state matrices as a whole, rather than focusing on individual optimization goals, e.g. the entropy of the duration matrix. Behavioral features encode aspects of the decision-making process over time, thereby loosely linking scheduling algorithm behavior and scheduling problem. An example of this is the number of trivial decisions when a single operation can be chosen from the resource buffer to the total number of sequencing decisions until the current state.

Additionally, within the general categories, features are further separated by the most closely associated scheduling goals. The fact that structural and behavioral features have no associated goal category is indicated by the presence of "None" in the associated goal column.

The formal definitions from Table 6.1 are largely based on the tracker variables within FabricatioRL state. When introducing the employed trackers, we simplify the notation by not including WIP and time indices explicitly. While we do not explicitly distinguish between WIP indices and the global environment-state matrices within the below formulae, all the trackers calculated over job indices pertain solely to WIP jobs. This is required since information outside the WIP, in particular not yet released jobs, should not be leaked to scheduling algorithms. With respect to the time indices, the reader should also note, that the operation matrices O^P, O^{Ty}, O^D, the operation location matrix L and the operation status matrix S as well as the current resource c change with the system time. We differentiate between the feature computation scheme of resource- and job-centric features (i), structural features (ii), and behavioral features (iii):

(i) Eleven trackers are needed for the *definition of resource- and job-centric features*.The number of completed operations per job is monitored by the variables $T_j^{co} := o_j - \mathbb{1}_{\{O_{ji}^{Ty} != 0\}}$, where o_j represents the original number of operations in job j. The total number of completed WIP operations T^{co} can be then defined as $T^{co} := \sum_j T_j^{co}$. Similarly, we define the number of remaining operations per job T_j^{ro} as $o_j - T_j^{co}$ and the total number of remaining WIP operations T^{ro} as $\sum_j T_j^{ro}$. $T_i^{Bfl} := \sum_{j,l} \mathbb{1}_{\{(L_{jl}=i) \wedge (S_{jl}=1)\}}$ represents the length of the buffer associated with the machine l. Using the remaining work trackers for $T_j^{rw} := \sum_i O_{ji}^D$ where j is a WIP job, we define the total amount of remaining processing time (work) over all WIP operations as $T^{rw} := \sum_{j \in \mathcal{J}} T_j^{rw}$. $T_j^R := t - R_j$ tracks the time elapsed since the release of job j. For some features, we will need the job operation processing times at the time of the job arrival. We use variables $T_j^{iw} := \sum O_{ji}^{Ty,R_j}$ to denote these quantities. Building on the previous trackers we define the work done variables for jobs j as $T_j^{dw} := T_j^{wi} - T_j^{rw}$. The buffer load of resource i with respect to the total processing time of the contained operations is given by $T_i^{Bft} := \sum_{j,l} \mathbb{1}_{\{(L_{jl}=l) \wedge (S_{jl}=1)\}} \cdot D_{jl}$. Note that $S_{jl} == 1$ if and only if the corresponding operation l from job j is buffered at the resource L_{jl}.

(ii) To *define our structural features*, we use a series of metrics that quantify the heterogeneity of jobs in terms of their duration and types. To this end, we first employ two editing distance operators namely the Hamming Distance H (Warps, 1983) and Kendal Tau τ (Kendall, 1938). We apply τ to individual rows from the O^{Ty} using the identical permutation as a reference vector. Secondly, we cluster jobs with respect to their types and duration using k-means and calculate the silhouette coefficient of the resulting clusters for three values of k namely 2, $w-1$ and $\lfloor \frac{w}{2} \rfloor$. Here w represents the WIP size. Before applying k-means we use min-max scaling on the type and duration matrix rows and concatenate the matrices column-wise. We use S_k to denote the application of the clustering scheme to our data. Last but not least, we use the categorical cross-entropy operator C over the flattened type and duration matrices to extract additional numeric values for the matrix entry heterogeneity as a whole. The cross-entropy is defined as $\sum p_i \log(p_i)$, where p_i is the probability of occurrence of category i. Applied to our matrices i translates to the discrete types for O^{Ty} and to one of 10 quantized duration values for O^D.

(iii) The variables needed for all the *behavioral feature definitions*, save for the Heuristic Agreement Entropy (F46) are the WIP index set $\mathcal{W}$, the arrived job set $\mathcal{K}$, the set of discrete decision time-points $\mathcal{T}$ and the legal action set $\mathcal{L}$. These state-dependent sets are maintained by the employed simulation, similar to the previously introduced trackers. The last section of Table 6.1 uses the sets above to define the behavioral features in a straightforward way. F46's definition requires additional elucidation. We first run the ten sequencing heuristics $H_k, k \in \{1, \ldots 10\}$ from Section 5.1.1 and note down the array of selected operations $(o_{H_k})_{k \in \{1,\ldots,10\}}$. Note that two heuristics could yield the same operation index. We then compute the relative frequencies $p_i^{o_{H_k}}$ of the distinct operation indices i within the o_{H_k} array. Finally, the entropy over these relative frequencies is calculated yielding our feature.

TABLE 6.1: FEATURE OVERVIEW

Nr	**Name**	**Definition**	***FJc* Rank**	***Jm* Rank**	**Related Goal**	**Category**
$F1$	Buffer Length Ratio	$T_c^{Bfl} \cdot (T^{ro})^{-1}$	43	40	Bf_i/Bft_i	Resource-Centric
$F2$	Buffer Load Avg.	$\mu_i(T_i^{Bft} \cdot (\max_{i \in \mathcal{M}} T_i^{Bft})^{-1})$	23	37		
$F3$	Buffer Load St.D.	$\sigma_i(T_i^{Bft} \cdot (\max_{i \in \mathcal{M}} T_i^{Bft})^{-1})$	22	26		
$F4$	Buffer Time Ratio	$T_c^{Bft} \cdot (T^{rw})^{-1}$	11	29		
$F5$	Current Utilization	$m^{-1} \cdot \sum_{ji} \mathbb{1}_{\{S_{ji}=2\}}$	48	42	Utl_i	
$F6$	Utilization Avg.	$\mu_i(Utl_i)$	12	**2**		
$F7$	Utilization St.D.	$\sigma_i(Utl_i)$	16	**7**		
$F8$	Estimated Tardiness Rate	$(T^{ro})^{-1} \cdot \sum_{ji} \mathbb{1}_{\{(t+\sum_i O_{ji}^D)>T_j\}}$	37	36	T_j	

Continued ...

TABLE 6.1: FEATURE OVERVIEW (Continued)

Nr	Name	Definition	***FJc*** **Rank**	***Jm*** **Rank**	**Related Goal**	**Category**
$F9$	Tardiness Rate	$(T^{ro})^{-1} \cdot \sum_{ji} \mathbb{1}_{\{(\sum_i O^D_{ji})>T_j\}}$	**7**	35	T_j	
$F10$	Operation Completion Rate	$T^{co} \cdot (\sum_j o_j)^{-1}$	45	45	Tpt_t	
$F11$	Work Completion Rate	$T^{do} \cdot (\sum_j T^{iw}_j)^{-1}$	21	28		
$F12$	Job Operation Completion Rate Avg.	$\mu_j(T^{co}_j \cdot o_j^{-1})$	46	46		
$F13$	Job Operation Completion Rate St.D.	$\sigma_j(T^{co}_j \cdot o_j^{-1})$	14	39		
$F14$	Job Operation Max Relative Completion Rate Avg.	$\mu_j(T^{co}_j \cdot (\max_{j \in \mathcal{J}} T^{co}_j)^{-1})$	25	43		
$F15$	Job Operation Max Relative Completion Rate St.D.	$\sigma_j(T^{co}_j \cdot (\max_{j \in \mathcal{J}} T^{co}_j)^{-1})$	**4**	38		
$F16$	Job Work Completion Rate Avg.	$\mu_j(T^{dw}_j \cdot (T^{iw}_j)^{-1})$	42	33		
$F17$	Job Work Completion Rate St.D.	$\sigma_j(T^{dw}_j \cdot (T^{iw}_j)^{-1})$	**8**	27		
$F18$	Max Relative Work Completion Rate Avg.	$\mu_j(T^{dw}_j \cdot (\max_{j \in \mathcal{J}} T^{dw}_j)^{-1})$	18	18	F_j	Job-Centric
$F19$	Max Relative Work Completion Rate St.D.	$\sigma_j(T^{dw}_j \cdot (\max_{j \in \mathcal{J}} T^{dw}_j)^{-1})$	**5**	17		
$F20$	Absolute Job Throughput Time Avg.	$\mu_j(T^{dw}_j \cdot (\max_j R_j)^{-1})$	13	**10**		
$F21$	Absolute Job Throughput Time St.D.	$\sigma_j(T^{dw}_j \cdot (\max_j R_j)^{-1})$	17	15		
$F22$	Relative Job Throughput Time Avg.	$\mu_j(T^{dw}_j \cdot R_j^{-1})$	**9**	20		
$F23$	Relative Job Throughput Time St.D.	$\sigma_j(T^{dw}_j \cdot R_j^{-1})$	**3**	16		
$F24$	Estimated Flow Time Avg.	$\mu_j(T^R_j + T^{rw}_j \cdot \frac{1}{m} \sum_{i \in \mathcal{M}} Utl_i)$	**10**	24		
$F25$	Estimated Flow Time St.D.	$\sigma_j(T^R_j + T^{rw}_j \cdot \frac{1}{m} \sum_{i \in \mathcal{M}} Utl_i)$	**1**	19		
$F26$	Makespan Lower Bound to Upper Bound Ratio	$\frac{1}{m} \cdot (F17 + T^{wr}) \cdot (F17 + T^{wr})^{-1}$	36	23	$C_{\max}$	
$F27$	Normalized Duration Avg.	$\mu_{j,i}((O^D_{ji} - O^D_{\min}) \cdot (O^D_{\max} - O^D_{\min})^{-1})$	**7**	35		
$F28$	Normalized Duration St.D.	$\sigma_{j,i}((O^D_{ji} - O^D_{\min}) \cdot (O^D_{\max} - O^D_{\min})^{-1})$	**2**	21		
$F29$	WIP Relative System Time	$t - \min_j R_j$	15	11		
$F30$	Normalized Type Avg.	$\mu_{j,i}((O^{Ty}_{ji} - O^{Ty}_{\min}) \cdot (O^{Ty}_{\max} - O^D_{\min})^{-1}))$	32	34		
$F31$	Normalized Type St.D.	$\sigma_{j,i}((O^{Ty}_{ji} - O^{Ty}_{\min})) \cdot (O^{Ty}_{\max} - O^{Ty}_{\min})^{-1})$	33	25	None	Structural
$F32$	Duration Distance Mean	$\mu_{j,k}(H(O^D_j, O^D_k))$	39	31		
$F33$	Duration Distance St.D.	$\sigma_{j,k}(H(O^D_j, O^D_k))$	20	**8**		
$F34$	Type Hamming Mean	$\mu_{j,k}(H(O^{Ty}_j, O^D_k))$	41	30		
$F35$	Type Hamming St.D.	$\sigma_{j,k}(H(O^{Ty}_j, O^D_k))$	27	13		
$F36$	Kendall Tau Avg.	$\mu_j(\tau(O^{Ty}_j))$	40	**6**		

Continued ...

TABLE 6.1: FEATURE OVERVIEW (Continued)

Nr	**Name**	**Definition**	**FJc Rank**	**Jm Rank**	**Related Goal**	**Category**
$F37$	Kendall Tau St.D.	$\sigma_j(\tau(O^{Ty}_j)$	35	14	None	Structural
$F38$	Silhouette Minimum k	$S_2(O^{Ty}, O^D)$	26	**9**		
$F39$	Silhouette Mid k	$S_{\lfloor w/2 \rfloor}(O^{Ty}, O^D)$	31	22		
$F40$	Silhouette Maximum k	$S_{w-1}(O^{Ty}, O^D)$	38	44		
$F41$	Type Entropy	$C(O^{Ty})$	29	12		
$F42$	Duration Entropy	$C(O^D)$	34	**4**		
$F43$	WIP to Arrival Ratio	$\lvert\mathcal{K}\rvert \cdot \lvert\mathcal{W}\rvert^{-1}$	44	47	None	Behavioral
$F44$	WIP to Arrival Time Ratio	$(\sum_{j \in \mathcal{K}} \sum_{i < o_j} O^D_{ji}) \cdot (\sum_{j \in \mathcal{W}} \sum_{i < o_j} O^D_{ji})^{-1}$	28	49		
$F45$	Decision Skip Ratio	$\sum_{t \in \mathcal{T}} \mathbb{1}_{\{T^{Bfl}_{c(S_t)}=1\}} \cdot \lvert\mathcal{T}\rvert^{-1}$	19	**1**		
$F46$	Heuristic Agreement Entropy	$\sum_i p_i^{o_{H_k}} \log(p_i^{o_{H_k}})$	47	41		
$F47$	Legal Action to Job Ratio	$\lvert\mathcal{L}\rvert \cdot \lvert\mathcal{W}\rvert^{-1}$	49	48		
$F48$	Legal Action Length Stream Avg.	$\mu_{t \in \mathcal{T}}(\lvert\mathcal{L}_t\rvert)$	30	**3**		
$F49$	Legal Action Length Stream St.D.	$\sigma_{t \in \mathcal{T}}(\lvert\mathcal{L}_t\rvert)$	**6**	**5**		

We normalized our features to fit in the $[0, 1]$ interval to avoid implicitly biasing our models towards features with larger magnitudes and to speed up training. There are few exceptions to this rule, e.g. the WIP to arrival ratio $F43$, whose values lie within the $[0, 2]$ interval. Non-normalized featured are due to either desiring values more easy to interpret for humans (the exemplified WIP to arrival ratio is useful for understanding system behavior — see Figure 6.5 and Figure 6.6), or due to no obvious constant time normalization being possible (e.g. duration entropy).

The top ten ranking induced by the feature_importance property of the Random Forest regressor trained to predict the makespan achieved by SimSearch given the described features differs between FJc and Jm instances:

FJc — The ten best features within the FJc setup are, in order of their importance,

1 the Estimated Flow Time Standard Deviation ($F25$),

2 the Normalized Duration Standard Deviation ($F28$),

3 the Relative Job Throughput Time Standard Deviation ($F23$),

4 the Job Operation Max Relative Completion Rate Standard Deviation ($F15$),

5 the Max Relative Work Completion Rate Standard Deviation ($F19$),

6 the Legal Action Length Stream Standard Deviation ($F49$),

7 the Normalized Duration Average ($F27$),

8 the Job Work Completion Rate Standard Deviation ($F17$),

9 the Relative Job Throughput Time Average ($F22$), and

10 the Estimated Flow Time Average ($F24$).

This amounts to one behavioral features ($F49$), two structural features ($F28$, $F27$), and seven job-centric features ($F15, 17, 19, F22 - F25$).

Jm — In the Jm case, the ten best features are

1 the Decision Skip Ratio ($F45$),

2 the Utilization Average ($F6$),

3 the Legal Action Length Stream Average ($F48$),

4 the Duration Entropy ($F42$),

5 the Legal Action Length Stream Standard Deviation (F49),

6 the Kendall Tau Average ($F36$),

7 the Utilization Standard Deviation ($F7$),

8 the Duration Distance Standard Deviation ($F33$),

9 the Silhouette Minimum k ($F38$) and

10 the Absolute Job Throughput Time Average ($F20$).

This feature set has a very different categorial composition as compared to the FJc feature set. It contains three behavioral features ($F45$, $F48 - 49$), four structural features ($F33$, $F36$, $F38$, $F42$) two recource-centric features ($F6 - 7$) and a single job-centric feature ($F20$).

Three observations can be made based on the above feature ranking. First, the presence of structural and behavioral features in the top ten features for both setups — $3/10$ in the FJc case and $7/10$ in the Jm case — confirms the relevance of our novel features. Secondly, encoding related information in different ways does not necessarily constitute a redundancy. This is suggested by the fact that all the job-centric features in the top ten of the FJc setup are associated with the flow-time goal. Finally, the thin overlap in top ten features between the two setups, as well a the vast difference in categorial composition, foreshadows the limitations in transfer capabilities for models trained on one setup in terms of performance on the other.

6.2.4 Action Design

In terms of the action-space, we investigate two direct (D1, D2) and one indirect action designs (D3) from literature. In the D3 design, the agent chooses between one of six simple heuristics, which in turn select the operation to be set to processing on the resource currently requesting a task.

Direct Actions D1-D2: Recall that the direct actions are interpreted differently between the two designs:

D1 In D1 our agents need to indicate the operation for the now free machine by means of a WIP operation index. The vector of possible actions corresponds in shape to the flattened operation matrix, limited to 14 rows corresponding to the current WIP jobs. The training environment then internally unravels the agent action into a tuple, which is used to implement the simulation logic. To the best of our understanding, the works of Jiménez (2012), Mendez-Hernandez et al. (2019), Waschneck et al. (2018), Thomas et al. (2018), and Rinciog et al. (2020) model actions in a similar fashion.

D2 Within the D2 action design, agents select a job index from the set present in WIP rather than an operation directly. Since both FJc and Jm have linear precedence graphs, the job index uniquely identifies the next job operation. Similar approaches were taken by Gabel et al. (2007a), Gabel (2009), Gabel et al. (2012), Reyna et al. (2015), Fonseca-Reyna et al. (2018), Zhang et al. (2020), and Park et al. (2021).

Note that in the raw action modeling case, agents can take illegal actions (see Figure 6.8), which would lead to a training environment reset. In fact, in most cases, the majority of the DQN output vector will correspond to illegal actions. Having many possible *illegal actions can significatly slow down or even impede agent learning*, since RL algorithms must first learn the "rules of the game".

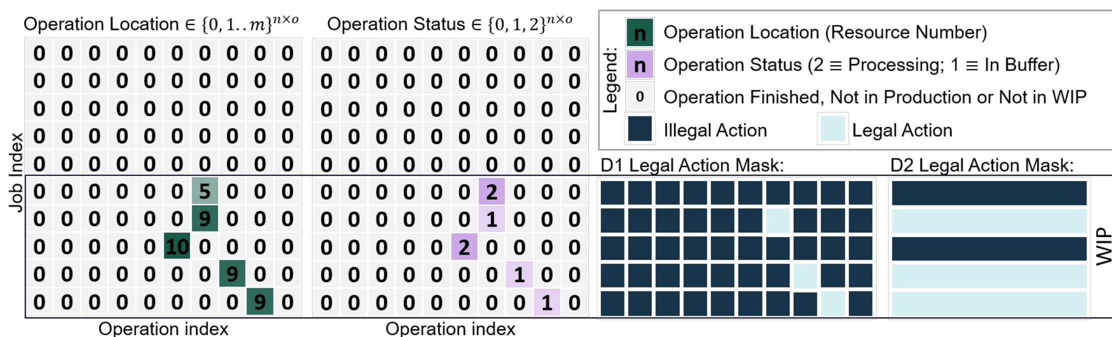


Figure 6.8: Illegal action masking for the direct action design from an environment perspective. Assume that machine $c = 9$ requests a decision. The viable operation indices (j, i) are those corresponding to operations present at the target resource ($L_{ji} = c$ in the matrix on the left) with a status indicating in buffer wait ($S_{ji} = 1$ in the matrix on the right). In D1 only three out of 50 actions are legal: $\mathcal{L}_{D1} := \{(j, i) \mid L_{ji} = c \wedge S_{ji} = 1\}$, i.e. they do not lead to an environment reset. In D2 only the job index needs to be specified, leading to three out of five possible actions being legal: $\mathcal{L}_{D2} := \{j \mid \exists i : L_{ji} = c \wedge S_{ji} = 1\}$. To prevent the agent from taking illegal actions, its output vector needs to be multiplied with the mask (bottom right in the figure). In case of NNs, the adjusted outputs need to be additionally considered during the backpropagation step.

Brammer et al. (2022) note, citing Zahavy et al. (2018) that "there is currently no straightforward way of preventing invalid actions in reinforcement learning". Nevertheless, there is an arguably straightforward way to overcome this caveat, namely by explicitly zeroing out the RL Agent's network output corresponding to illegal actions and re-normalizing the values prior to the final action selection. This has to be done both after a network forward

pass and before a network backward pass. Masking out illegal actions would correspond to endowing the agent with knowledge of the domain rules and was done in the case of AZ (see Section 4.2.2).

Figure 6.8 provides an example of how masks could be constructed using FabricatioRL, which supports this technique by means of the get_legal_actions API function. Legal actions in the D1 case correspond to operation indices (j, i) currently positioned at the resource requesting a sequencing decision, $c = 9$ in the figure, as indicated by the resource numbers in the location matrix ($L_{ji} = c$). FabricatioRL returns the sets $\mathcal{L}_{D1} := \{(j, i) \mid L_{ji} = c \wedge S_{ji} = 1\}$ on get_legal_actions calls. In the D2 case, the legal actions correspond to the set of corresponding job indices $\mathcal{L}_{D2} := \{j \mid \exists i : L_{ji} = c \wedge S_{ji} = 1\}$.

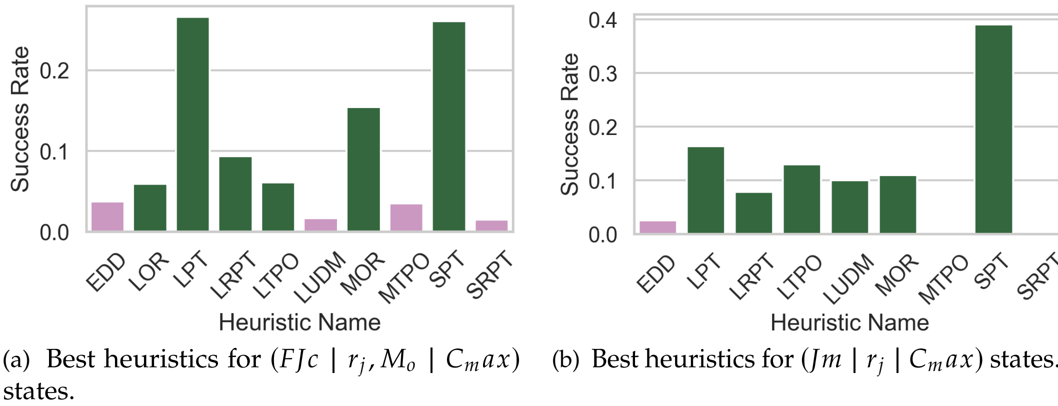
We do not employ masking within our DQN experiments for two reasons. Firstly, masking needs to be additionally supported by the RL agent implementation, since the outputs adjusted through masking should be considered during backpropagation. Currently, there is no widely spread RL agent repository implementing the technique for DQN. Secondly, there is no instance of such an approach being used with DQN in RL scheduling literature. The sole instance of masking being used in the context of the reviewed RL scheduling work, is proposed by Rinciog et al. (2020), where an unpublished implementation of AZ is used. Note that our AZ implementation also makes use of the described masking technique.

Indirect Actions D3: We do not use all ten heuristics present in the data accumulation loop for two reasons. On the one hand, we want to ease the agent's learning process, on the other hand, as can be seen from Figure 6.9a, there is a clear heuristic hierarchy for the FJc setup. The plot counts the number of times different heuristics achieved the best makespan among their peers, i.e. they "won" the scheduling game. We use this information to exclude heuristics that perform poorly from the agent's action-space. The figure isolates EDD, LUDM, MTPO, and SRPT as the exclusion candidates since these simple heuristics win, in less than 5% of the cases accumulated as described in Section 6.1.2.

Figure 6.9b shows the distribution of the most successful heuristics for the Jm data accumulated. We can clearly see a discrepancy between the heuristic performance of the two plots. This discrepancy is yet another indicator of the potentially limited transfer potential between the two setups.

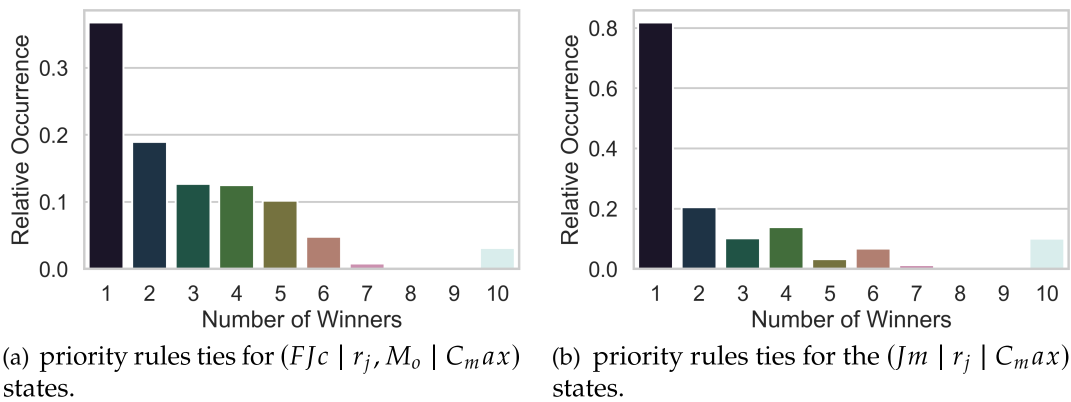
Another indicator foreshadowing poor transfer learning results is depicted in Figure 6.10. The figures count the number of times multiple priority rules lead to the same winning makespan result for the FJc and Jm states respectively. The difference in tie distribution between the FJc and Jm setups suggests a different, setup-dependent pattern.

Additionally, the tie distributions give insight into the D3 learning task difficulty. The large number of ties ($\approx 40\%$ for FJc) suggests that the task may be (very) difficult for our models, since the mapping of states to best priority rules is often not unique, and thus, not a function.



(a) Best heuristics for $(FJc \mid r_j, M_o \mid C_max)$ states.
(b) Best heuristics for $(Jm \mid r_j \mid C_max)$ states.

Figure 6.9: Distribution of the state-dependent best heuristics for the considered setups setup. The distribution is calculated over 127,922 and 189,901 data points in the Fjc and Jm cases respectively. The magenta bars represent heuristics that achieve the best results in under 5% of the considered cases.



(a) priority rules ties for $(FJc \mid r_j, M_o \mid C_max)$ states.
(b) priority rules ties for the $(Jm \mid r_j \mid C_max)$ states.

Figure 6.10: Distribution of the priority rule ties within the context of simulation search. The figure counts the relative occurrence of multiple priority rules being tied for the first place in terms of makespan for 127,922 FJc states (a) and 189,901 Jm states (b). Such ties could confuse D3 RL models. The different distribution of ties between setups points towards difficulties in transfer learning.

6.3 Training and Evaluation

Supplementary to the considerable number of design choices available to us when considering RL applications for production scheduling, deep-learning also presents the hurdle of a large (hyper-)parameter space. Aside from the parameters defining neural network architecture, key hyper-parameters associated with both neural networks, e.g. learning rate, regularization, and RL, e.g. discount factor, exploration parameters, have to be considered.[1] Furthermore, the learning process is in-transparent, with no obvious correlation between loss decrease and model performance.

After a concise introduction of the experiment setup in Section 6.3.1, we detail our

[1]The distinction between parameters and hyper-parameters for learning algorithms is given by the nature of their relationship with the training data. If the model variable are inferred from data during training, e.g. NN weights, then the variable is a parameter. Otherwise the model variable is a hyper-parameter. (Kuhn et al., 2013)

experimental results in two phases. First, we evaluate RL models resulting from different parametrizations and designs against each other in Section 6.3.2. After isolating the most promising design using DQN we additionally train an AZ to establish a comparison between different RL paradigms. Then, in Section 6.3.3 we deploy the best DQN model together with AZ model to large FJc and Jm test sets. The added Jm test set serves to evaluate the limits of learning transfer between significantly different scheduling setups. We discus the scheduling results relative to the baselines introduced in Chapter 5.

6.3.1 Experiment Setup

The proposed designs are evaluated during the model selection phase using DQN agents. This is because of DQN's popularity in the RL production scheduling community. Having chosen the best design and hyper-parameter set, we then train AZ using the DQN results, deferring a dedicated AZ model selection phase to future work. Training both DDQN and AZ reveals AZ to be the more attractive choice, because of the fewer hyper-parameters required, and faster training speeds. The subsequent evaluation section further reinforces this, seen as AZ outperforms DDQN.

The model selection phase uses ten different seeds to generate ten different ($FJc \mid r_j, M_i^o | C\text{max}$) instances as described in Section 6.1.2 for agent training. FabricatioRL cycles through these instances on calls to reset.[2] For each of the three designs (D1, D2, D3) we additionally vary three learning rate values and two replay buffer size values. We train each of the resulting 18 models for one million steps, which, for the slowest design, D1, takes several days. The fastest design, D3, finishes training in less than a day. D1 and D2 models are trained in parallel two at a time using two GPUs. D3 models are trained six at a time, distributing three models on two GPUs with 11 GB of memory each. The single AZ is trained alone on the GPU with self-play episodes running concomitantly on the CPU. All training is monitored using the TensorBoard.

For the model evaluation we sample 6000 seeds uniformly at random to generate corresponding ($FJc \mid r_j, M_i^o | C\text{max}$) instances on which we test the best DQN model, together with AZ and the RL-competitive baselines introduced in Chapter 5. Additionally, we also generate 6000 ($Jm \mid r_j | C\text{max}$) instances to which we deploy all the algorithms to be compared. This is done to test another sometimes postulated RL advantage, namely transfer learning. Different scheduling algorithms are evaluated with respect to winns and makespan relative to the virtual best selector, i.e. the ex-post best algorithm for every instance. Job routing decisions are taken by all algorithms except CP3 using the LQT heuristic. Using LQT and CP3 in conjunction is difficult if at all possible, because of the different paradigms they represent (planning vs reactive scheduling), and beyond the scope of this work.

All models except for DQN are ran on the CPU during evaluation. DQN running on a GPU if available is the standard configuration in Stable-Baselines 3 with which we chose not to interfere. To speed up the process, we use 40 different threads, varying seeds between

[2]These are triggered by the agent in the background on simulation completion. A simulation run terminates either if all jobs have been processed, or if an illegal action was suggested by the agent.

them. In particular all the compared algorithms run in (the same) sequence in all threads. All experiments were ran on a server with two 11GB GPUs, an IntelXeon processor with 24 cores and 64GB of RAM. The available hardware allowed for the partial parallelization of both the model selection and the model evaluation phase, as previously described. However, with the exception of AZ, the individual algorithms used throughout this paper are single threaded.

6.3.2 Model Selection

This section will demonstrate the difficulties of training the popular DQN approach found in RL production scheduling literature. Additionally, we will demonstrate the implausibility of the direct action designs D1 and D2, in the absence of legal action masking, for large production scheduling instances. While our designs are informed by choices encountered in literature, our experiments show that DQN agents fail to learn the "the rules of the game", thus leaving the D3 design as the only viable option.

DQN Parameters: For all our designs we use fully connected NN architectures with Rectified Linear Unit (RELU) activation functions for all hidden layers. The activation function of the output layer is the identity function, as needed by the DQN algorithm. For D1 and D2 we use three fully connected hidden layers of sizes 256, 512, and 256. For D1, where the input has 564 entries and the output vector has a length of 140, this amounts to a total of 432,640 parameters. Using the same architecture for D2 yields 400384 parameters, since in this case, the network output has a size of 14. We set the network size using the RL production scheduling literature as an orientation, e.g. 320-256-128-128-13 (Hofmann et al., 2020) or 512-128-18 (Waschneck et al., 2018). Also guided by the literature, e.g. 64-32-16-8 (Park et al., 2019) or 100 (Stricker et al., 2018), we use a considerably smaller network for our feature state indirect action design (D3). Here the agent network consists of a single hidden layer with 128 nodes, yielding a total of 2182 parameters (10 inputs and 6 nodes in the output layer).

The DQN flavor we chose from Stable-Baselines 3 is a DDQN with a history replay buffer and soft target updates. This introduces several additional parameters on top of the learning rate α, exploration rate ϵ, the ϵ-decay, the reward discount factor γ, and the total number of data points to gather, i.e. the total number of environment steps to take before learning halts. As stated in Section 4.2.1, DDQN uses a less frequently updated target network $f_{\theta'}$ to construct the Q-values f_θ is trained on. The target network updates consist of re-initializing θ' using θ for which the frequency and the soft update parameter τ have to be set.

It would go beyond the scope of this work to test the influence of each individual parameter on the overall agent learning behavior and its associated performance. As such most of the DDQN parameters are set to fixed values. We do, however, investigate the impact of the learning rate and buffer size during model selection. For α we chose three initial values namely $1e{-3}$, $1e{-4}$ and $1e{-5}$. These values are plugged into a linear learning rate schedule (Darken et al., 1990) which gradually decreases the learning rate such that it reaches 0 just after the final gradient update. For β we used two values, namely $1e4$ and

$1e5$. Table 6.2 gives an overview of all DDQN parameter values, both fixed and variable. Variable parameters are placed within accolades.

Table 6.2: The values taken by the DDQN parameters during model selection.

Name	Symbol	Values		
		D1	D2	D3
Self-Play/Target Network	$f_\theta / f_{\hat{\theta}}$	FCNN: 256-512-256		FCNN: 128
Learning Rate	α	$\{1e{-}3, 1e{-}4, 1e{-}5\}$		
Buffer Size	β	$\{1e4, 1e5\}$		
Batch Size	b	128	$2e3$	
Polyak Update Coefficient	τ	0.8		
Discount factor	γ	0.99		
Initial Exploration Rate	ϵ	0.5		
ϵ-Decay	ϵ_{decay}	33%		
Minimum Exploration Rate	ϵ_{min}	0.01		
Training Frequency	$k_{\text{self-play}}$	$1e3$		
Target Update Interval	k_{target}	$1e3$	$2e3$	

We use an initial ϵ of 0.5 with an ϵ-decay calculated such that the final ϵ value of 0.01 is reached after 33% of the total training steps ($1e6$ for D1-2 and $2 \cdot 1e6$ for D3). This implicit definition of the ϵ-decay is a Stable-Baselines 3 implementation flavor. The employed γ has a value of 0.99 thereby clearly prioritizing the immediate reward over future rewards. We do this since the training environment skips over trivial decisions leading to subsequent states, and therefore rewards, that are sufficiently far in the future by design.

The update of the action selection network f_θ is performed every 1000 steps (training frequency) on batch sizes of 128 for the D1 setup and 2000 on the D2 and D3 setups. For D1 we sample a single batch from the replay buffer to perform the weight update, while for D2 and D3 we train on the entirety of data in the replay buffer. We perform fewer gradient steps on a much smaller batch in the D1 case to save time, given that we expect the agent to frequently pick illegal actions, which leads to a significant overhead induced by the training environment re-initialization. In the D1 experiment setup we set the batch size to a conservative 128 whereas in D2 and D3 we increased this value to 2000.

The soft update parameter is assigned a value of 0.8 and $f_{\theta'}$ is updated every 2000 steps. Lastly, note the first f_θ update is performed at step 2000 and that the replay buffer is a FIFO queue, meaning that the oldest experience is pushed out of the buffer when its maximal capacity β is reached.

To obtain a better view of the model performance evolution during the exploration stages ($\epsilon > 0.01$), we use the evaluation episode parametrization implemented in Stable-Baselines 3. This amounts to testing the agent in a purely exploitative fashion ($\epsilon = 0$) for a fixed number of episodes after a fixed number of steps. In our case, "episode" refers to scheduling all the 100 jobs in our scheduling instances to completion, or until an illegal

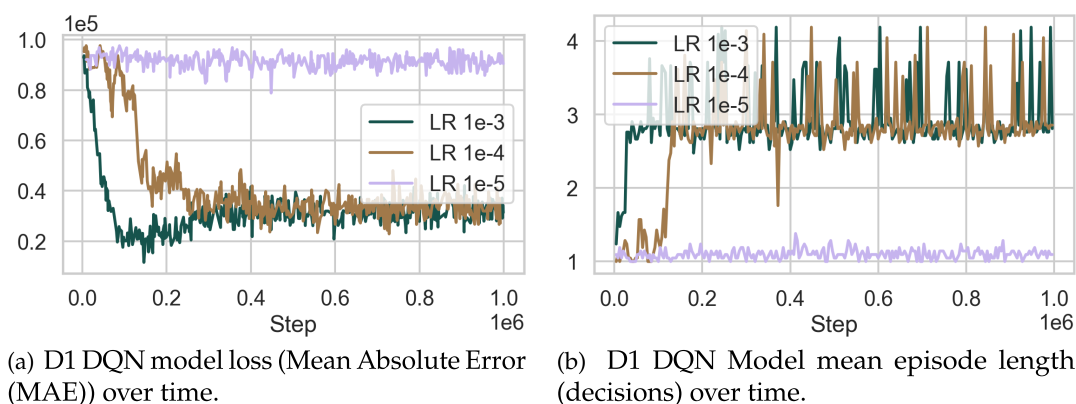
action occurs. We evaluate our agents every 4000 steps on 20 differently seeded scheduling episodes. For agent training monitoring, we log the f_θ loss, i.e. MSE, along with the mean episode reward, mean episode length as well as the progression of the learning rate and exploration rate. The mean episode reward and the episode length signals are monitored separately for training and evaluation.

DQN Model Selection D1-D2: Since the direct action setups can yield illegal actions we adapted the makespan reward to severely punish agents in the case of illegal actions. The resulting reward $R'_6(S_n)$ administers punishment inversely proportional to the number of legal actions taken until state nth state as defined by Equation 6.12, where $t(S_n)$ denotes the system time associated with state S_n, $\mathcal{L}_{D1/2}(S_n)$ the corresponding legal action set, and a_n the action leading to the reward.

$$R'_6(S_n) := \begin{cases} \min(-2e4, -1e3 \cdot (100 - n)), & \text{if } a_n \notin \mathcal{L}_{D1/2}(S_n) \\ R_6(S_t), & \text{otherwise} \end{cases} \tag{6.12}$$

The learning curves associated with both D1 and D2 agent show that the algorithms struggle to learn the "rules of the game":

D1 Figure 6.11 displays the D1 model learning curves for the three different learning rates and a replay buffer size $\beta = 1e4$. Looking solely at the training loss in Figure 6.11a we see that the agents do discern a pattern given a high enough initial learning rate. However, the discerned pattern is not sufficient for learning what the legal scheduling actions are for all given states. As can be seen from Figure 6.11b, which depicts the average number of steps per episode, agents only pick legal sequencing actions an average of four times before the environment resets. This is less than 5% of a normal episode length which oscillates around 70 steps for the FJc case. We notice that an initial learning rate of $1e{-}5$ leads to agents not being able to learn anything (the loss in Figure 6.11a does not decrease over time). Both $1e{-}4$ and $1e{-}3$, however, determine a similar loss convergence and legal action selection performance.

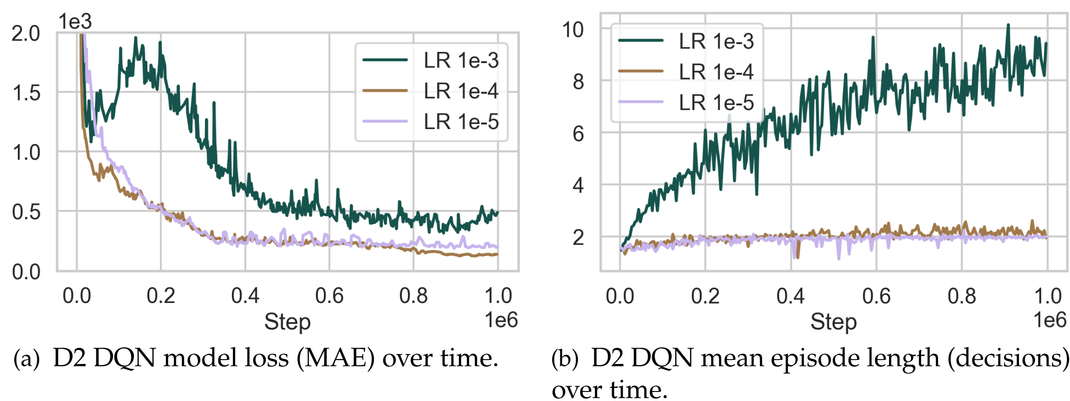


(a) D1 DQN model loss (Mean Absolute Error (MAE)) over time.

(b) D1 DQN Model mean episode length (decisions) over time.

Figure 6.11: Learning for the raw state direct (operation) action models using a replay buffer size $\beta = 1e4$. "LR" designates the model learning rate. While the models pick up a pattern for larger learning rates (left), they cannot infer the "rules of the game" (right).

D2 For the D2 design, the agents display a similar, albeit somewhat more successful learning behavior. Figure 6.12 displays the training loss and mean evaluation episode length plotted against the number of training steps for the agents with the D2 design trained using a buffer size of $1e4$. Here the best agent learns to pick a legal action in about 15% of the cases. Considering that the worst case probability of picking illegal actions is $1 - \frac{2}{14} = 0.86$ for D2 and $1 - \frac{2}{140} = 0.99$ it is unsurprising that the D2 models fare better as compared to the D1 models. All agents converge towards a loss of around 500 for all learning rates, with the lower learning rates displaying less noise and a slightly better convergence behavior. However, while the models trained with initial learning rates of $1e{-}4$ and $1e{-}5$ have a better convergence behavior (i.e. less noise and a smaller convergence error), than their $1e{-}3$ counterpart, they perform considerably worse in terms of legal action selection. Thus it can be theorized, that RL learning curves indicate whether the model has found a consistent strategy rather than a good one. Note that the D2 model trained with the $1e - 3$ seems to keep improving at the end of $1e6$ steps



(a) D2 DQN model loss (MAE) over time.

(b) D2 DQN mean episode length (decisions) over time.

Figure 6.12: Learning curves for the raw state direct (job) action models using a replay buffer size $\beta = 1e4$. "LR" deisgnates the model learning rate. While the models pick up a pattern for larger learning rates (left), they cannot infer the "rules of the game" (right). The model with the highest learning rate ($1e{-}3$) fares best and its training is not complete after 1e6 steps.

The following two conclusions ca be drawn from the displayed learning behavior:

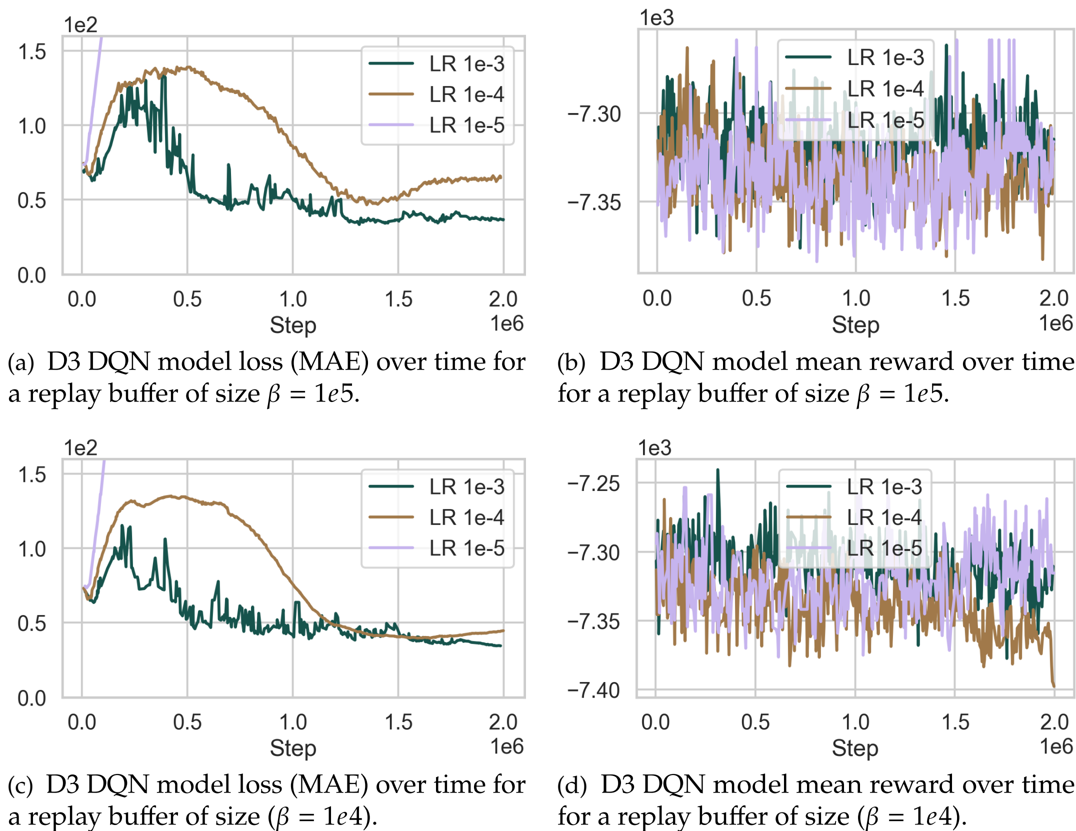
1. The learning rate significantly impacts the agent's learning behavior as can be seen from both D1 and D2 experiments. Moreover, the learning rate impact is different depending on the chosen design.

2. There seems to be no general correlation between training loss and model performance. This effect is exemplified by the learning curves in Figure 6.12.

While our agents' learning failure is obvious, the reasons for it are less transparent. This is because there is no way to determine whether the model under- or over-fits on the training data from a single training run. Underfitting can happen if the model is not strong enough to fit the data in the replay buffer. In such a case we would need to add more parameters to the agent's network, increase the learning rate, decrease the buffer size or add more

gradient steps. Conversely, a non-converging loss can occur because of overfitting. In this case, the agent perfectly fits the data currently in the replay buffer. As soon as new data is added, the error shoots up again because of the model's poor generalization capacity. In such a case we would need to increase the training buffer size, decrease the number of nodes in the network, decrease the learning rate or decrease the number of gradient steps. In our case, the poor model performance is most likely due to under-fitting, since the D1 and D2 experiments ran with the larger buffer size ($1e5$) display a very similar learning behavior.

DQN Model Selection D3: The indirect action experiments show more promise. On the one hand, agents following the D3 design, need not be concerned with discerning legal from illegal actions, which makes the learning task more manageable. On the other hand, learning is faster so we can more easily train for more steps. Figure 6.13 shows the loss and mean evaluation episode reward for the models trained with a buffer size of $1e4$ and $1e5$ respectively over a span of $2e6$ steps.



(a) D3 DQN model loss (MAE) over time for a replay buffer of size $\beta = 1e5$.

(b) D3 DQN model mean reward over time for a replay buffer of size $\beta = 1e5$.

(c) D3 DQN model loss (MAE) over time for a replay buffer of size ($\beta = 1e4$).

(d) D3 DQN model mean reward over time for a replay buffer of size ($\beta = 1e4$).

Figure 6.13: D3 model learning curves in terms of MAE loss (left subplots) and average episode reward (right subplots) for replay buffers of size $\beta = 1e5$ (top) and $\beta = 1e4$ (bottom). "LR" designates the model learning rate. The models with a learning rate of $1e{-}5$ diverge. Average episode rewards are noisy and do not display an improving trend.

The following four observations can be made. Firstly, the models trained over larger replay buffers achieve slightly better training losses than the ones trained with a smaller replay buffer. Secondly, the initial learning rate of $1e{-}5$ is too small and leads to divergence. Thirdly, the model loss gets worse before it gets better. This may be due to the smaller

amounts of data in the replay buffer, in the beginning, coupled with a high exploration rate. As old data starts being expelled from the buffer and the action selection scheme starts to gradually favor exploitation, the model loss starts decreasing again. Fourthly, there is no stable pattern of increasing reward accumulation with an increase in training steps. Rather the mean reward achieved during the evaluation stage oscillates around -7375 over the entire training period.

Overall, our best DQN candidate is the D3 agent with a buffer size of $1e5$ and a learning rate of $1e3$. The D1 and D2 agents can be excluded from the contest since they did not learn to discern legal from illegal actions. Note, however, that one could evaluate these agents as well, by using the masking technique previously discussed during deployment. From the batch of D3 agents, we picked the ones trained with $1e5$ reply buffer sizes, since their overall training losses are slightly better, signaling more confidence in their own strategy. Of these three models, we chose the one trained with a learning rate of $1e3$ because this agent has both the best loss after $2e6$ steps and the highest peak during evaluation with an average cumulative makespan reward of -7241.

AZ Parameters: We use the D3 design and the best DQN learning rate to train an AZ agent. Save for the learning rate and buffer sizes, AZ shares no further parameters with DQN. Since, as opposed to DQN, the AZ network is trained on the experience buffer in its entirety (see Section 4.2.2), we employ the smaller buffer size of $1e4$ to save training time. The AZ own parameter v_{puct}, temperature τ and number of MCTS iterations per step are set to 2.5, 1 and 2 during training. During the later evaluation, we increased the number of iterations per step to 5. Table 6.3 gives an overview of the parameter values including the essential details of the model's NN.

Table 6.3: The values taken by the AZ parameters during model selection.

Name	**Symbol**	**Value**
Policy-Value Network	f_θ	Shared Stack: Conv(64,3,3)-Id(64,3,3)x2 π-Head: Conv(2,1,1) v-Head: Conv(2,1,1)-Dense(64)
Learning Rate	α	1e−4
Buffer Size	β	1e4
Number MCTS Iterations	$i_{\max}$	2
Puct Paramer	c_{puct}	2.5
Temperature	τ	1

In terms of the AZ implementation, we use a small version of the ResNet architecture. While the feature vector inputs of D3 do not display spatial correlations, temporal correlations may be useful for learning (see the feature evolution in Annex C). We use a stack of three consecutive state feature vectors as the network input. After an initial convolutional layer, we pass our input through three identity blocks (consisting of the main path and a skip connection) consisting of two convolutional layers on the main path. All convolutional layers up to this point consist of 64 filters with a kernel size of 3×3, a stride of 1, and padding that maintains the shape integrity of the input. The policy head applies the

last convolution with two filters a kernel size of 1 and a stride of 1 thereby condensing the depth of the input to 2 activation maps. These are then concatenated and passed through a dense layer of the action-space dimension with a softmax activation function. The value head distinguishes itself from the policy head by adding a dense layer with 64 neurons between the 1× convolutional layer and the single-neuron output layer. All network neurons have a $RELU$ activation function and batch normalization is applied to all convolutional layer outputs.

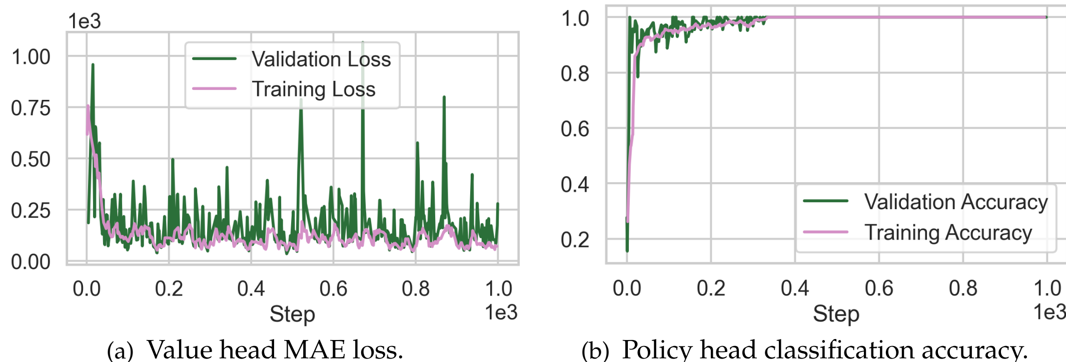
Compared to DQN, AZ performs fewer scheduling steps but requires a comparable time for training in spite of the parallel self-play and network update processes. The exact number of AZ training steps is difficult to determine since our implementation is parameterized by episodes, which are set to $1e4$. FJc instances require an average of 70 sequencing decisions to finish. Hence, the total of $1e4$ episodes of self-play we employed amounts to approximately $6e5$ steps, which is an order of magnitude lower than the number of steps used for DQN.

AZ Model Training: Our AZ agent learning process is split into 500 iterations of interlaced self-play and network training. During self-play 20 episodes are run in parallel on the CPU with the accumulated experiences being added to the replay buffer at the conclusion of the last scheduling game. Meanwhile, the agent network is being trained on the GPU using the experience buffer of the previous iteration. We do two passes over the entire replay buffer during every network training step. This amounts to $1e3$ network updates. Since data points in the replay buffer get pushed out by new experiences approximately every ten episodes, the network uses every data point during a backpropagation step around $1e2$ times.

The AZ learning curves shown in Figure 6.14 suggest that our agent is all but certain of his strategy. Figure 6.14a shows the value head MAE over the $1e3$ backpropagation steps while Figure 6.14b displays the policy head accuracy. Since the buffer content of two consecutive training iterations is 80% identical, we use 10% of the buffer contents to plot the validation loss at each backpropagation step, so as to get a better feel for the model generalization behavior. The figure shows that both network heads generalize quite well on the training episode data, with the value head displaying more of an inclination toward overfitting. While the value head still misses the actual state value by around 125, the policy head displays absolute confidence in what the selected strategy should be (accuracy of near 100% with respect to the MCTS selected action).

6.3.3 Evaluation

To ease the comparability of the scheduling algorithms at hand, we make use of the Virtual Best Selector (VBS) concept (Lindauer et al., 2015). The VBS is the ex-post best scheduling algorithm for any particular instance with respect to makespan. Let $\mathcal{A}$ be the set of scheduling algorithms investigated, i.e. the ten simple heuristics along the best DQN model, AZ, CP3, and our simulation-based approaches SimSearch and MCTS. The VBS score for the instance generated using the seed n_i is where algorithms $A \in \mathcal{A}$ have achieved makespan scores of $C^A_{\max}(n_i)$ is defined in Equation 6.13. The average VBS relative



(a) Value head MAE loss. (b) Policy head classification accuracy.

Figure 6.14: AZ learning curves. The training was stopped prematurely since the model seemed to have converged.

makespan scores of the compared scheduling algorithms $A \in \mathcal{A}$ are then computed as defined by Equation 6.14, where i takes the value of the 6000 seeds used.

$$C_{\max}^{VBS}(n_i) := \min_{A \in \mathcal{A}} C_{\max}^{A}(n_i) \tag{6.13}$$

$$C_{\max}^{A^{VBS}} := \frac{\sum_i C_{\max}^{VBS}(n_i)}{\sum_i C_{\max}^{A}(n_i)} \tag{6.14}$$

The parameter sets used for the different algorithms have mostly been elucidated. Nevertheless, we briefly review them here. For the DDQN and AZ the parameter sets in Table 6.2 and 6.3 were used. The snapshot-based approaches are non-parametric. Note that the maximum number of operations to be considered by the CP solver (see Equation 5.11), is in fact an approach parameter. To keep with the frame of this work, however, we kept it fixed to three as reflected by our naming (CP3).

The most important simulation-based approach parameters are represented by the employed priority rules. SimSearch runs with all ten sequencing heuristics from Table 5.1 and the LQT and LQO job routing priority rules. Its completion percentage parameter p (see Section 5.2.1) is set to 0.6. Furthermore, to save time, for the FJc case only, the roll-outs are only executed every ten decisions. In-between the last winning priority rules is used as to select the next action. MCTS is ran using heuristic actions corresponding to the same priority rules used by SimSearch. On every decision, the selection, expansion, roll-out and backtracking steps are executed eight times. The UCT constant C is set to two.

The algorithms are compared with respect to two aspects. First, we look at the runtime of the different algorithm categories (1). Secondly, to emphasize the different picture painted by different metrics, we look at the scheduling performance (2) with respect to the number of "wins", i.e. the number of instances where the scheduling approach is equal to the VBS and at the average VBS-relative makespans achieved by our scheduling methods. To illustrate the importance of a solid baseline, we first compare RL approaches solely against simple priority rules as is often done in literature, which may give the impression that RL is a very good contender. We then add CP3 and our RL-competitive baselines to the mix.

When considering all baselines, we first look at the CP3-DQN-AZ-best heuristic group and then at the extended group additionally containing SimSearch and MCTS.

(1) **Runtime Discussion:** With respect to runtime (Figure 6.15), the conversation is quite straightforward albeit important. All considered scheduling methods are appropriate for deployment in an online context judging by the average experiment runtime values and the little variance thereof. The longest recorded runtimes are associated, unsurprisingly, with MCTS on the FJc setup, with an average of 1228 seconds being required for a complete scheduling run, which consists of around 200 decisions (100 operations á two decisions each). Hence the time required for a single decision is around 6.14 seconds (recall that the routing decisions were fixed to LQT), which should be sufficient for most scheduling cases. For RL algorithms and priority rules, the number of decisions is less than half (60 on average), because of the fixed job routing priority rules and FabricatioRL's decision skip mechanism (see Section 3.2.3).

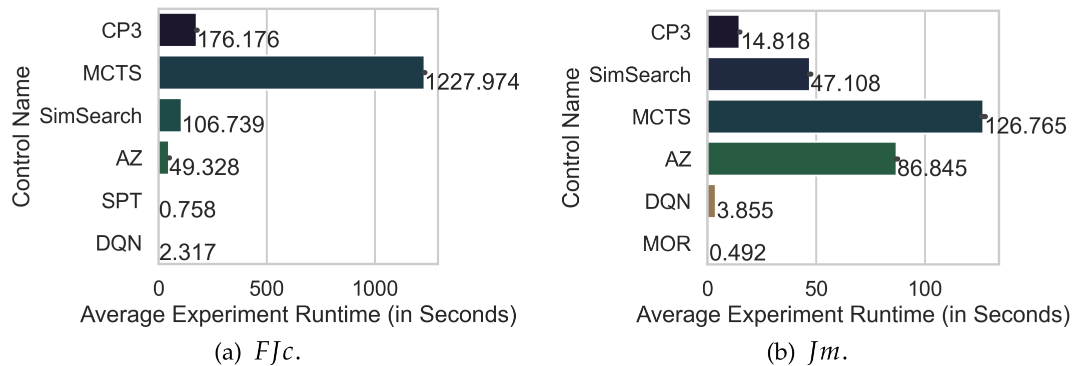


Figure 6.15: Average runtime for scheduling approaches for a full instance scheduling.

Note that the hierarchy of runtimes also differs between setups. In the FJc case, MCTS is slowest with 1228 seconds per instance, followed by CP3 with 176 seconds, AZ with 49 seconds per instance, DQN with 2 seconds per instance, and SPT which needs less than one second for a complete instance run. In the Jm case, MCTS is slowest needing 127 seconds, followed by AZ with 87 seconds per instance, CP3 with 15 seconds, DQN with 4, and heuristic approaches needing about half a second.

The general decrease in runtimes (with the exception of DQN and AZ) from the FJc setup to the Jm in spite of more decisions being required on average (around 100) for the Jm setup may be surprising at a first glance. However, the explanation is simple. The reason is the higher degree of flexibility of the FJc setup. The lack of flexibility makes solving CP instances much simpler, hence the radical runtime decrease of CP3 by more than 90%. Another important factor here is the system being underbooked, meaning that fewer WIP operations need to be considered by CP3 on average.

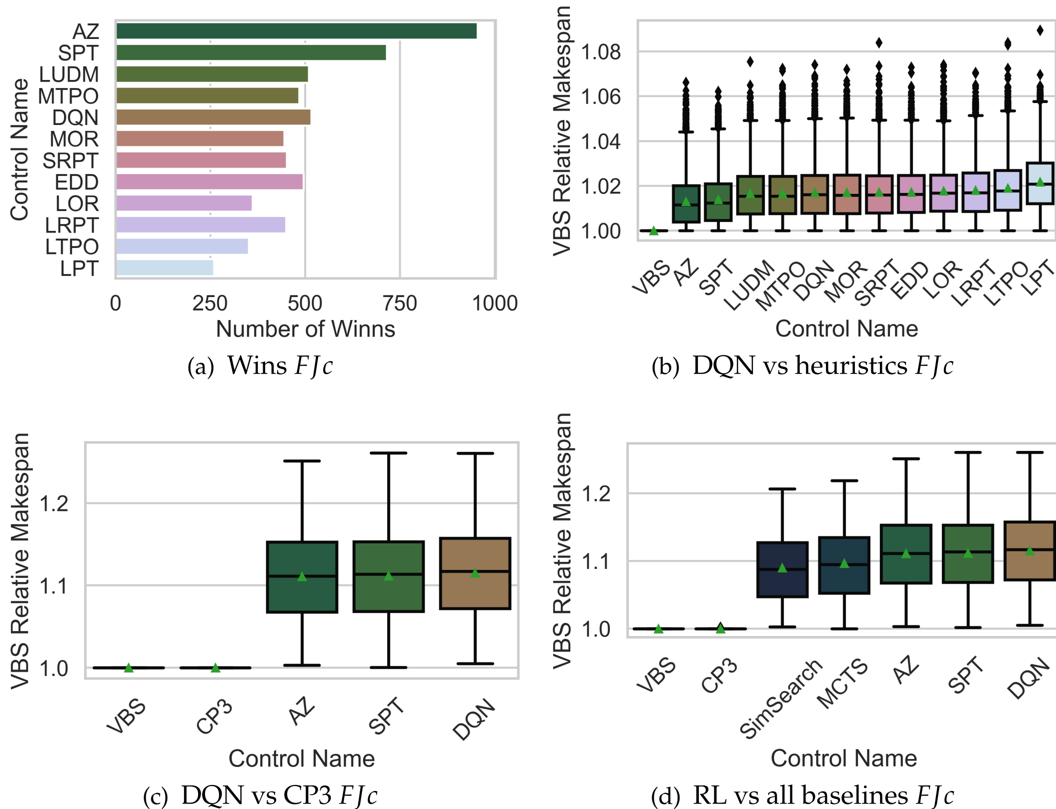
Note that an important overhead associated with RL is the switch between NN calls to predict individual states. NN frameworks optimize for arrays of data being processed all at once rather than individual data points. If a GPU is used for `predict` calls, as is the case for our DQN application, there is the additional overhead of

moving data from RAM to GPU memory. This coupled with the larger number of decisions in the Jm case could explain why the average DQN runtime increases by 1.5 seconds moving from FJc to Jm.

(2) ***FJc* Performance Discussion:** The results for the FJc setup experiment illustrated by Figure 6.16 three observations, two pertaining to the scheduling hierarchy and two pertaining to the comparison approach. Note that the first two subfigures compare scheduling approaches excluding CP3.

AZ is the best scheduling method with respect to the number of wins (around 900 out of 6000 evaluation instances — Figure 6.16a) and VBS relative score (Figure 6.16b). In terms of the latter indicator, however, AZ only slightly outperforms the best simple heuristic, namely SPT (by less than one percent point). The DQN performance is quite poor. Our best DQN model is the third best in terms of wins (≈500/6000), but it is overshadowed by AZ and even SPT (≈700/6000) (Figure 6.16a). In terms of relative makespan, it ranks fifth (Figure 6.16a).

When introducing CP3 into the mix (Figure 6.16c), the new baseline is indistinguishable from the VBS and significantly better than both RL approaches and the best FJc heuristic, SPT. Finally, from Figure 6.16d we see that the simulation-based baselines outperform AZ, but do not fare so well when compared to CP3. This is most likely due to LQT, the job routing priority rule used by both SimSearch and MCTS, not being sufficiently strong. Note however, that both algorithms could use a set of heuristics instead of one.



(a) Wins FJc

(b) DQN vs heuristics FJc

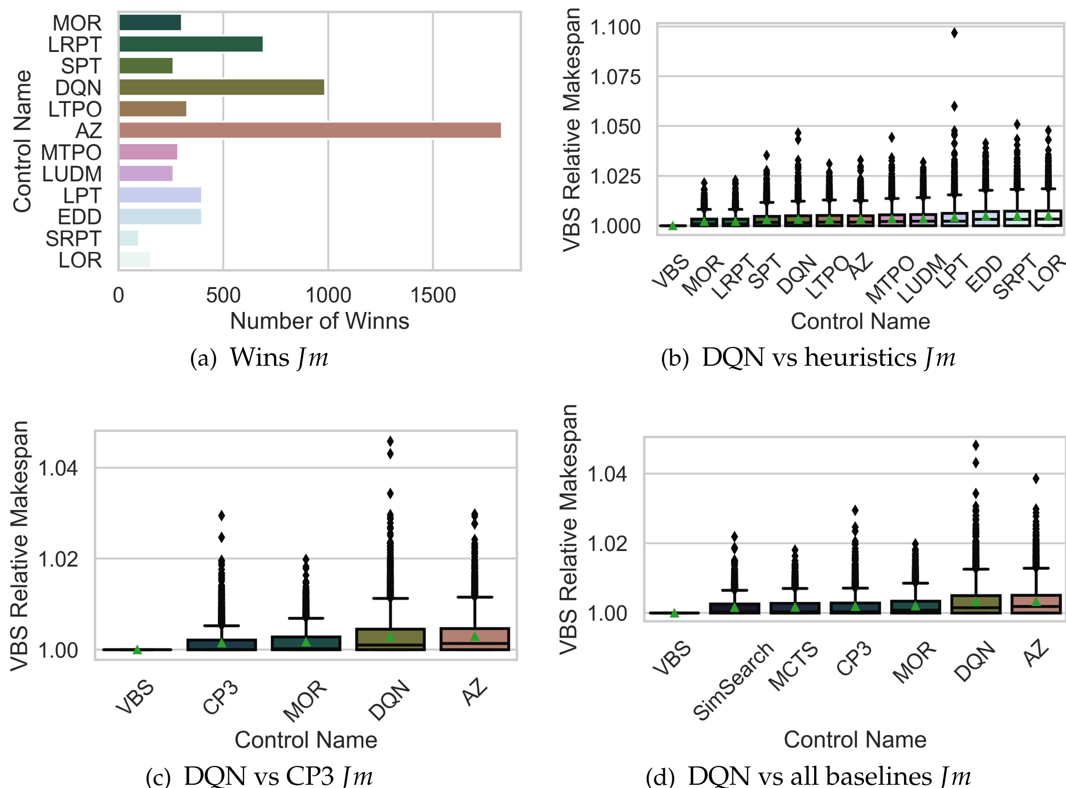
(c) DQN vs CP3 FJc

(d) RL vs all baselines FJc

Figure 6.16: FJc results.

(2) ***Jm* Performance Discussion (Transferability):** The results for the Jm setup testing the RL model transferability are shown in Figure 6.17. Similarly to the FJc results, the first two subfigures exclude CP3 from the comparison. We observe the following: In terms of wins, the RL algorithms are top tier occupying the first two positions in the scheduling approach ranking (Figure 6.17a). Once again AZ seems to outperform the other approaches with approximately 1700 wins within the 6000 instances. With respect to VBS relative makespan, the hierarchy drastically changes, with RL approaches displaying mediocre performances (Figure 6.17b). The top four positions are occupied by four heuristics among which MOR is the strongest and DQN this time marginally outperforms AZ.

When including CP3 in Figure 6.17c, the added baseline still outperforms the other approaches, albeit, this time, not by a large margin. Last but certainly not least, when additionally considering our simulation-based approaches (Figure 6.17), we notice that CP3 stops being the best solution. Since in the Jm case job routing priority rules are not required, this could be indicative of the relative strength of our sequencing priority rules. Note however, that the difference in performance between all the considered algorithms is only very slight.



(a) Wins Jm

(b) DQN vs heuristics Jm

(c) DQN vs CP3 Jm

(d) DQN vs all baselines Jm

Figure 6.17: Jm results.

The different viewpoints of our experiment results serve to strengthen the reader's impression that the evaluation approach matters a lot when discussing (stochastic) scheduling results. Firstly, there is the matter of chosen evaluation metric. If we were to focus solely on wins, for instance, we would give the impression that the RL approaches,

Table 6.4: The makespan achieved by the different scheduling algorithms in the FJc and Jm setups. The table rows are sorted ascendingly by the FJc results. CP3 significantly outperforms all other approaches.

Rank	Control	FJc
1	VBS	6543.362 (-0.000%)
2	CP3	6543.376 (-0.000%)
3	SimSearch	7117.735 (-8.070%)
4	MCTS	7158.987 (-8.599%)
6	AZ	7252.663 (-9.780%)
7	SPT	7257.879 (-9.845%)
8	LUDM	7277.637 (-10.089%)
9	MTPO	7277.757 (-10.091%)
10	DQN	7280.755 (-10.128%)
11	MOR	7280.815 (-10.129%)
12	SRPT	7281.439 (-10.136%)
13	EDD	7281.779 (-10.141%)
14	LOR	7285.697 (-10.189%)
15	LRPT	7287.514 (-10.211%)
16	LTPO	7294.237 (-10.294%)
17	LPT	7314.608 (-10.544%)
18	RND1	8805.107 (-25.687%)
19	RND2	8992.310 (-27.234%)

Rank	Control	Jm
1	VBS	11574.535 (-0.000%)
2	SimSearch	11598.260 (-0.205%)
3	MCTS	11599.297 (-0.213%)
4	CP3	11600.112 (-0.220%)
5	MOR	11602.956 (-0.245%)
6	LRPT	11603.182 (-0.247%)
8	SPT	11613.443 (-0.335%)
9	DQN	11615.028 (-0.349%)
10	LTPO	11615.835 (-0.356%)
11	AZ	11615.912 (-0.356%)
12	RND1	11617.816 (-0.373%)
13	MTPO	11618.321 (-0.377%)
14	LUDM	11618.944 (-0.382%)
15	RND2	11621.819 (-0.407%)
16	LPT	11625.108 (-0.435%)
17	EDD	11631.796 (-0.492%)
18	SRPT	11632.794 (-0.501%)
19	LOR	11634.392 (-0.514%)

particularly AZ, perform much better than they actually do. Though it stands to reason, that approaches outperforming others in many situations are better, the winning margin is even more important, which is why measuring the optimization target directly should be preferred. Secondly, the chosen baselines can leave the reader with vastly different impressions with respect to the efficacy of the baselined approach. From 6.16b it looks as if there are three tiers of scheduling approaches, with AZ and SPT being in the first tier, DQN in the second, and LPT in the last. Conversely, baselining against CP3 only makes AZ, DQN, and SPT look almost the same in terms of performance. Thirdly, the idea of picking priority rules dependent on the production state seems to be justified, since both AZ and the simulation-based approaches fare better than all the other heuristics in the FJc case. However, given that both MCTS and SimSearch outperform AZ, simulation seems to be a more reliable strategy than prediction.

Inter-Experiment Discussion: For a more fine-grained view of the numeric results, we also provide the average makespan scores over the 6000 scheduling instances along with the distance of individual scores from the VBS in percent in Table 6.4. The results in the table are sorted ascendingly by the FJc average makespan scores. Notice that we included two flavors of random heuristics in the listed results. As the name suggests, these heuristics simply pick a random operation from the resource buffer (RND1), or randomly pick between buffered operations and a wait signal (RND2). In the FJc case, these heuristics fare very poorly, whereas, in the Jm case, their performance is only mediocre.

Overall, there is a significant shift in results between the FJc and Jm setups. Most notably

these differences pertain to the following four aspects.

i RL Approach Rank Shift: The decrease in AZ model performance is clear, which points towards a limited transferability capacity. However, this is not surprising, given what we know about the significantly different scheduling setup behaviors. DQN holds fast to its mediocrity, which makes it difficult for us to infer its transferability potential.

ii Result outliers: The magnitude and number of outliers within the Jm VBS relative makespan results are much higher than in the FJc setup. This points to more randomness in the composition of the scheduling instances.

iii Score tightness: The scheduling algorithm score tightness in the Jm case is much higher than in the FJc case. This points to the more limited optimization potential of the Jm setup. If the myopic approaches (simple heuristics) perform similarly to approaches with a wider planning horizon (CP3), we can expect more planning would not necessarily lead to better performance.

iv Heuristic Ranking: The lack of distinction between heuristics approaches in particular, can point either to randomness or limited optimization potential. The employed baseline heuristics, albeit simple, are quite diverse in terms of the strategic approach to scheduling. Recurring patterns within scheduling instances (less randomness) would lead to a more stable heuristic hierarchy. Additionally, all heuristics lead to the same outcome for many individual decision points during the scheduling run, e.g. because the vast majority of decisions are trivial, there is most likely not much that a less myopic approach could improve.

Establishing the optimization potential of a scheduling problem is key to selecting an appropriate scheduling method. It is in great part because of this aspect, that the scheduling setup behavior (Section 6.1.2) should be carefully considered. Note at this point, that there is a close relationship between randomness and optimization potential: A high degree of randomness surely limits the available optimization potential. The lack of the latter, however, does not necessarily imply the former. A scheduling setup where all decisions are trivial, for instance, has no optimization potential but can be perfectly deterministic. The lack of optimization potential within the Jm can have various causes. It could be that the job pool is not diverse enough, the system does not offer much in the way of flexibility (e.g. no job routing flexibility), or the system is underbooked. Both latter factors could explain the result tightness and number of outliers in the Jm setup.

Chapter 7

Conclusion and Future Work

People do not like to think. If one thinks, one must reach conclusions. Conclusions are not always pleasant.

— Helen Keller

In this work, we sought to bring order to the growing yet unstructured body of work that is RL production scheduling, thereby lifting some of the mysticism surrounding it. We achieved our goal by first creating a standardization framework for experimental work in the field. We then implemented a benchmarking simulation framework that allows for the reproducible execution of the vast majority of the experiments in literature. To eliminate some of the positive bias towards RL, we chose and developed strong scheduling approaches sharing some of the advantages of RL to serve as competitive baselines. Finally we ran a series of experiments on two popular stochastic setups, namely $(FJc|r_{ji}, M_i^o|C_{\max})$ and $(FJc|r_{ji}, M_i^o|C_{\max})$, pitting DDQN and AZ against our baselines.

Our experimental scheduling method comparison yielded the following direct conclusions which point to the overarching conclusion that *RL is not a jack of all trades*. First, DQN, which is by far the most popular RL approach in the literature, struggles to outperform even simple heuristics. AZ marginally succeeds in this endeavor. Secondly, the RL learning transferability between problems is poor, at least in cases such as ours, where problem attributes, particularly flexibility and system load, are significantly different. Thirdly, the proposed baselines, namely CP3, SimSearch and MCTS offer a good challenge for RL approaches, seen as they outperformed the priority rules and RL approaches on both the FJc and the Jm setups. The performance difference was significant for the former and marginal for the latter.

The current elaboration served to relativize the advantages of RL within the context of production scheduling to some extent. On the one hand we selected, adapted or developed algorithms which share the advantages of RL. On the other hand, the model selection process makes it clear that training RL approaches is an uphill battle because of the many design options and model parameters, the long training times, and the difficulty of establishing the link between learning progress and performance increase.

Table 7.1: Perceived RL solution approach advantages compared to exact re-planning and priority rules.

Criteria	Priority Rules	RL	SimSearch	MCTS	CP3	Exact Re-Planning
Deterministic Setup Solution Quality	Medium	High	High	High	High	Optimal
Adaptivity	High	High	High	High	High	Low
Transferability	Low	Low	High	High	Medium	Medium
Runtime Efficiency	High	High	High	High	High	Low
Mathematical Modeling Overhead	Low	Low	Low	Low	High	High
Design and Parameter Tuning Overhead	Low	High	Low	Low	Low	Low
Simulation Engineering Overhead	Low	High	High	High	Low	Low

Furthermore, building the simulation which is a necessary condition for RL model training adds supplementary overhead. Table 7.1 compares our baselines with RL showcasing these conclusions.

In trying to understand the reasons behind the different scheduling algorithm behavior between setups we noted how relevant setup transparency is. In particular, scheduling setup clarity allows us to create sound hypotheses about the scheduling problem optimization potential and the influence of stochasticity and flexibility on the results.

Much remains to be done in terms of future work. Aside from maintaining the proposed standardization framework and extending our benchmarking simulation framework — FabricatioRL — to cover at least all the experiments encountered in literature, and investigating the new baseline algorithms more deeply, we note the following four aspects derived from our experimental evaluation.

First, we should extend the setup analysis and create new benchmark sets varying stochasticity, and flexibility and noting down the optimization potential. Assessing these variables is nontrivial since no such indicators are known for the production scheduling case. Hence, this avenue of research also implies the development of such indicators.

Secondly, more RL studies need to be conducted on the setups introduced. While AZ, the best RL approach in this work in terms of scheduling performance, did not manage to outperform our baselines, that in no way means that RL in general cannot. Since the current results show that AZ has more potential than DQN, an in-depth look at AZ's performance employing other RL designs is needed.

Thirdly, we should even out the playing field between RL and CP3 by using RL to make job routing decisions as well as sequencing decisions. This can be done using either a single agent for both or dedicated RL agents for the individual sub-problems.

Finally, we need to experiment with different optimization goals. Makespan is easier for exact (re-planning) approaches such as CP to handle. Conversely, it is more difficult to solve scheduling problems with tardiness as a goal. This could further level the playing field in favor of RL helping us to isolate the situations where RL best fits in the field of production scheduling.

Bibliography

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/ (cit. on pp. 99, 103).

Adams, Joseph, Egon Balas, and Daniel Zawack (1988). "The shifting bottleneck procedure for job shop scheduling". In: *Management science* 34.3, pp. 391–401 (cit. on p. 12).

Aho, Alfred V., Michael R Garey, and Jeffrey D. Ullman (1972). "The transitive reduction of a directed graph". In: *SIAM Journal on Computing* 1.2, pp. 131–137 (cit. on p. 68).

Applegate, David and William Cook (1991). "A computational study of the job-shop scheduling problem". In: *ORSA Journal on computing* 3.2, pp. 149–156 (cit. on p. 12).

Armstrong, Eddie, Michele Garraffa, Barry O'Sullivan, and Helmut Simonis (2021). "The hybrid flexible flowshop with transportation times". In: *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (cit. on p. 14).

Arviv, Kfir, Helman Stern, and Yael Edan (2016). "Collaborative Reinforcement Learning for a Two-Robot Job Transfer Flow-Shop Scheduling Problem". In: *International Journal of Production Research* 54.4, pp. 1196–1209. DOI: 10.1080/00207543.2015.1057297 (cit. on pp. 20, 30, 38, 42, 51, IV, VIII, XI).

Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer (2002). "Finite-time analysis of the multiarmed bandit problem". In: *Machine learning* 47.2-3, pp. 235–256 (cit. on p. 117).

Aydin, ME and E Oztemel (2000). "Dynamic Job-Shop Scheduling Using Reinforcement Learning Agents". In: *Robotics and Autonomous Systems* 33.2-3, pp. 169–178. DOI: 10.1016/S0921-8890(00)00087-7 (cit. on pp. 29, 35, 60, 135, V, IX, XII).

Baer, Schirin, Jupiter Bakakeu, Richard Meyes, and Tobias Meisen (2019). "Multi-Agent Reinforcement Learning for Job Shop Scheduling in Flexible Manufacturing Systems". In: *2019 Second International Conference on Artificial Intelligence for Industries (AI4I 2019)*, pp. 22–25. DOI: 10.1109/AI4I46381.2019.00014 (cit. on pp. 88, IV, VIII, XI).

Baker, Kenneth R (2014). "Minimizing earliness and tardiness costs in stochastic scheduling". In: *European Journal of Operational Research* 236.2, pp. 445–452 (cit. on p. 24).

Barhebwa-Mushamuka, Felicien, Stéphane Dauzère-Pérès, and Claude Yugma (2019). "Work-in-process balancing control in global fab scheduling for semiconductor manufacturing". In: *2019 Winter Simulation Conference (WSC)*. IEEE, pp. 2257–2268 (cit. on p. 25).

Barnes, JW and JB Chambers (1996). "Flexible job shop scheduling by tabu search". In: *Graduate Program in Operations and Industrial Engineering, The University of Texas at Austin, Technical Report Series, ORP96-09* (cit. on p. 12).

Bartz-Beielstein, Thomas, Carola Doerr, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, Manuel Lopez-Ibanez, Katherine M Malan, Jason H Moore, et al. (2020). "Benchmarking in optimization: Best practice and open issues". In: *arXiv preprint arXiv:2007.03488* (cit. on p. 11).

Beasley, John E (1990). "OR-Library: distributing test problems by electronic mail". In: *Journal of the operational research society* 41.11, pp. 1069–1072 (cit. on pp. 12, 62, 66, 69, 124, 125).

Bell, Thomas E and Thomas A Thayer (1976). "Software requirements: Are they really a problem?" In: *Proceedings of the 2nd international conference on Software engineering*, pp. 61–68 (cit. on p. 54).

Benoit, Anne, Louis-Claude Canon, Redouane Elghazi, and Pierre-Cyrille Heam (2021). "Update on the Asymptotic Optimality of LPT". PhD thesis. Inria Grenoble-Rhône-Alpes (cit. on p. 83).

Bostock, Mike (2022). *D3.js - Data-Driven Documents*. URL: http://d3js.org/ (cit. on p. 80).

Bouazza, W., Y. Sallez, and B. Beldjilali (2017). "A Distributed Approach Solving Partially Flexible Job-Shop Scheduling Problem with a Q-Learning Effect". In: *IFAC Papersonline* 50.1, pp. 15890–15895. DOI: 10.1016/j.ifacol.2017.08.2354 (cit. on pp. 20, 25, 29, 42, VI, X, XIII).

Brammer, Janis, Bernhard Lutz, and Dirk Neumann (2022). "Permutation Flow Shop Scheduling with Multiple Lines and Demand Plans Using Reinforcement Learning". In: *European Journal of Operational Research* 299.1, pp. 75–86. DOI: 10.1016/j.ejor.2021.08.007 (cit. on pp. 20, 21, 45, 141, V, IX, XII).

Brandimarte, Paolo (1993). "Routing and scheduling in a flexible job shop by tabu search". In: *Annals of Operations research* 41.3, pp. 157–183 (cit. on p. 12).

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). *OpenAI Gym*. eprint: arXiv:1606.01540 (cit. on p. 4).

Browne, Cameron B, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton (2012). "A survey of monte carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1, pp. 1–43 (cit. on p. 117).

Bukkur, Khalid Muhamadin Mohamed Ahmed, MI Shukri, and Osama Mohammed Elmardi (2018). "A review for dynamic scheduling in manufacturing". In: *Global Journal of Research In Engineering* (cit. on pp. 27, 31).

Cai, Jingcao, Deming Lei, Jing Wang, and Lei Wang (2022). "A Novel Shuffled Frog-Leaping Algorithm with Reinforcement Learning for Distributed Assembly Hybrid Flow Shop Scheduling". In: *International Journal of Production Research*. DOI: 10.1080/00207543.2022.2031331 (cit. on pp. V, IX, XII).

Cao, ZhengCai, ChengRan Lin, and MengChu Zhou (2021). "A Knowledge-Based Cuckoo Search Algorithm To Schedule a Flexible Job Shop with Sequencing Flexibility". In: *IEEE Transactions on Automation Science and Engineering* 18.1, pp. 56–69. DOI: 10.1109/TASE.2019.2945717 (cit. on pp. 31, 37, 57, V, IX, XII).

Chen, Ronghua, Bo Yang, Shi Li, and Shilong Wang (2020). "A Self-Learning Genetic Algorithm Based on Reinforcement Learning for Flexible Job-Shop Scheduling Problem". In: *Computers&Industrial Engineering* 149. DOI: 10.1016/j.cie.2020.106778 (cit. on pp. IV, VIII, XII).

Chen, Xili, XinChang Hao, Hao Wen Lin, and Tomohiro Murata (2010). "Rule driven multi objective dynamic scheduling by data envelopment analysis and reinforcement learning". In: *2010 IEEE International Conference on Automation and Logistics*. IEEE, pp. 396–401 (cit. on pp. 25, 35, 43, 135, VI, IX, XIII).

Choo, H James (2017). "Technical Tutorial: Optimal Level of WIP in a Production System". In: *projectproduction.org* (cit. on pp. 25, 126).

Collobert, R., K. Kavukcuoglu, and C. Farabet (2011). "Torch7: A Matlab-like Environment for Machine Learning". In: *BigLearn, NIPS Workshop* (cit. on p. 99).

Coulom, Rémi (2006). "Efficient selectivity and backup operators in Monte-Carlo tree search". In: *International conference on computers and games*. Springer, pp. 72–83 (cit. on p. 118).

Csáji, Balázs Csanád (2001). "Approximation with artificial neural networks". In: *Faculty of Sciences, Etvs Lornd University, Hungary* 24, p. 48 (cit. on p. 93).

Csaji, BC, B Kadar, and L Monostori (2003). "Improving Multi-Agent Based Scheduling by Neurodynamic Programming". In: *Holonic and Multi-Agent Systems for Manufacturing*. Ed. by V Marik, D McFarlane, and P Valckenaers. Vol. 2744. Lecture Notes in Artificial Intelligence, pp. 110–123 (cit. on pp. IV, VIII, XI).

Cunha, Bruno, Ana Madureira, Benjamim Fonseca, and Joao Matos (2021). "Intelligent Scheduling with Reinforcement Learning". In: *Applied Sciences-Basel* 11.8. DOI: 10.3390/app11083710 (cit. on pp. 50, V, IX, XII).

Cunha, Bruno, Ana M Madureira, Benjamim Fonseca, and Duarte Coelho (2018). "Deep reinforcement learning as a job shop scheduling solver: A literature review". In: *International Conference on Hybrid Intelligent Systems*. Springer, pp. 350–359 (cit. on p. 12).

Dantsin, Evgeny, Vladik Kreinovich, and Alexander Wolpert (2022). "An AlphaZero-Inspired Approach to Solving Search Problems". In: *arXiv preprint arXiv:2207.00919* (cit. on p. 99).

Darken, Christian and John Moody (1990). "Note on learning rate schedules for stochastic optimization". In: *Advances in neural information processing systems* 3 (cit. on p. 145).

Dauzère-Pérès, Stéphane, W Roux, and Jean B Lasserre (1998). "Multi-resource shop scheduling with resource flexibility". In: *European Journal of Operational Research* 107.2, pp. 289–305 (cit. on p. 12).

Demirkol, Ebru, Sanjay Mehta, and Reha Uzsoy (1998). "Benchmarks for shop scheduling problems". In: *European Journal of Operational Research* 109.1, pp. 137–141 (cit. on p. 12).

Du, Yu, Jun-qing Li, Xiao-long Chen, Pei-yong Duan, and Quan-ke Pan (2022). "Knowledge-Based Reinforcement Learning and Estimation of Distribution Algorithm for Flexible Job Shop Scheduling Problem". In: *IEEE Transactions on Emerging Topics in Computational Intelligence*. DOI: 10.1109/TETCI.2022.3145706 (cit. on pp. 25, 30, V, IX, XII).

Durrett, Rick (2019). *Probability: theory and examples*. Vol. 49. Cambridge university press (cit. on p. 92).

Estivill-Castro, Vladmir and Derick Wood (1992). "A survey of adaptive sorting algorithms". In: *ACM Computing Surveys (CSUR)* 24.4, pp. 441–476 (cit. on p. 87).

Ey, H, D Sackmann, M Mutz, and J Sauer (2000). "Adaptive Job-Shop Scheduling with Routing and Sequencing Flexibility Using Expert Knowledge and Coloured PETRI Nets". In: *SMC 2000 Conference Proceedings: 2000 IEEE International Conference on Systems, Man&Cybernetics, Vol 1-5*. IEEE International Conference on Systems Man and Cybernetics Conference Proceedings, pp. 3212–3217 (cit. on p. 9).

Ferm, Tore and Albert Y Zomaya (2010). "A structured tabu search approach for scheduling in parallel computing systems". In: *Handbook of research on scalable computing technologies*. IGI Global, pp. 354–377 (cit. on p. 87).

Fisher, Henry (1963). "Probabilistic learning combinations of local job-shop scheduling rules". In: *Industrial scheduling*, pp. 225–251 (cit. on p. 12).

Fonseca-Reyna, Yunior César, Yailen Martínez-Jiménez, and Ann Nowé (2018). "Q-learning algorithm performance for m-machine, n-jobs flow shop scheduling problems to minimize makespan". In: *Investigación Operacional* 38.3, pp. 281–290 (cit. on pp. 36, 51, 141, IV, VIII, XI).

Gabel, Thomas (2009). "Multi-agent reinforcement learning approaches for distributed job-shop scheduling problems". In: *Osnabrük University Library* (cit. on pp. 34–36, 38, 48, 135, 141, IV, VIII, XI).

Gabel, Thomas and Martin Riedmiller (2007a). "On a successful application of multi-agent reinforcement learning to operations research benchmarks". In: *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. IEEE, pp. 68–75 (cit. on pp. 29, 34, 38, 45, 135, 141, IV, VIII, XI).

– (2007b). "Scaling Adaptive Agent-Based Reactive Job-Shop Scheduling To Large-Scale Problems". In: *2007 IEEE Symposium on Computational Intelligence in Scheduling*, pp. 259+. DOI: 10.1109/SCIS.2007.367699 (cit. on pp. 133, IV, VIII, XI).

– (2012). "Distributed Policy Search Reinforcement Learning for Job-Shop Scheduling Tasks". In: *International Journal of Production Research* 50.1, SI, pp. 41–61. DOI: 10.1080/00207543.2011.571443 (cit. on pp. 39, 51, 133, 141, VI, IX, XIII).

Gauci, Jason, Edoardo Conti, Yitao Liang, Kittipat Virochsiri, Zhengxing Chen, Yuchen He, Zachary Kaden, Vivek Narayanan, and Xiaohui Ye (2018). "Horizon: Facebook's Open Source Applied Reinforcement Learning Platform". In: *arXiv preprint arXiv:1811.00260* (cit. on p. 53).

Gelly, Sylvain and David Silver (2007). “Combining online and offline knowledge in UCT”. In: *Proceedings of the 24th international conference on Machine learning*. ACM, pp. 273–280 (cit. on p. 118).

Glasserman, Paul and David D Yao (1992). “Some guidelines and guarantees for common random numbers”. In: *Management Science* 38.6, pp. 884–908 (cit. on p. 62).

Glinz, Martin (2007). “On non-functional requirements”. In: *15th IEEE international requirements engineering conference (RE 2007)*. IEEE, pp. 21–26 (cit. on p. 54).

Graham, Ronald Lewis, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan (1979). “Optimization and approximation in deterministic sequencing and scheduling: a survey”. In: *Annals of discrete mathematics*. Vol. 5. Elsevier, pp. 287–326 (cit. on p. 11).

Grinberg, Miguel (2018). *Flask web development: developing web applications with python*. O’Reilly Media, Inc. (cit. on p. 80).

Gronauer, Sven and Klaus Diepold (2022). “Multi-agent deep reinforcement learning: a survey”. In: *Artificial Intelligence Review*, pp. 1–49 (cit. on p. 42).

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International Conference on Machine Learning*. PMLR, pp. 1861–1870 (cit. on p. 61).

Han, B. A. and J. J. Yang (2021). “A Deep Reinforcement Learning Based Solution for Flexible Job Shop Scheduling Problem”. In: *International Journal of Simulation Modelling* 20.2, pp. 375–386. DOI: 10.2507/IJSIMM20-2-CO7 (cit. on pp. 39, V, IX, XII).

Han, Bao-An and Jian-Jun Yang (2020). “Research on Adaptive Job Shop Scheduling Problems Based on Dueling Double DQN”. In: *IEEE Access* 8, pp. 186474–186495. DOI: 10.1109/ACCESS.2020.3029868 (cit. on pp. VII, X, XIII).

Han, Wei, Fang Guo, and Xichao Su (2019). “A Reinforcement Learning Method for a Hybrid Flow-Shop Scheduling Problem”. In: *Algorithms* 12.11. DOI: 10.3390/a12110222 (cit. on pp. 20, VI, X, XIII).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (cit. on p. 100).

Heger, Jens and Thomas Voss (2021). “Dynamically Adjusting the K-Values of the ATCS Rule in a Flexible Flow Shop Scenario with Reinforcement Learning”. In: *International Journal of Production Research*. DOI: 10.1080/00207543.2021.1943762 (cit. on pp. 31, 37, VII, X, XIV).

Henze, Norbert and Dieter Kadelka (2010). *Skript zur Vorlesung "Wahrscheinlichkeitstheorie und Statistik für Studierende der Informatik und des Ingenieurwesens"* (cit. on p. 92).

Hill, Ashley, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu (2018). “Stable Baselines”. In: *GitHub repository*. https://github.com/hill-a/stable-baselines (cit. on p. 53).

Hofmann, Constantin, Carmen Krahe, Nicole Stricker, and Gisela Lanza (2020). “Autonomous production control for matrix production based on deep Q-learning”. In: *Procedia CIRP* 88, pp. 25–30 (cit. on pp. 22, 51, 135, 145, VI, X, XIII).

Hong, JK and VV Prabhu (2004). "Distributed ReInforcement LearnIng Control for Batch SequencIng and SizIng in Just-In-time ManufacturIng Systems". In: *Applied Intelligence* 20.1, pp. 71–87. doi: 10.1023/B:APIN.0000011143.95085.74 (cit. on pp. IV, VIII, XI).

Hopp, Wallace J and Mark L Spearman (1991). "Throughput of a constant work in process manufacturing line subject to failures". In: *The International Journal Of Production Research* 29.3, pp. 635–655 (cit. on p. 25).

Hu, Hao, Xiaoliang Jia, Qixuan He, Shifeng Fu, and Kuo Liu (2020a). "Deep Reinforcement Learning Based AGVS Real-Time Scheduling with Mixed Rule for Flexible Shop Floor in Industry 4.0". In: *Computers&Industrial Engineering* 149. doi: 10.1016/j.cie.2020.106749 (cit. on pp. 29, VII, X, XIII).

Hu, Liang, Zhenyu Liu, Weifei Hu, Yueyang Wang, Jianrong Tan, and Fei Wu (2020b). "Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network". In: *Journal of Manufacturing Systems* 55, pp. 1–14 (cit. on pp. 51, 88).

Huang, George Q, YF Zhang, and PY Jiang (2008). "RFID-based wireless manufacturing for real-time management of job shop WIP inventories." In: *International Journal of Advanced Manufacturing Technology* 36 (cit. on p. 25).

Hubbs, Christian D, Hector D Perez, Owais Sarwar, Nikolaos V Sahinidis, Ignacio E Grossmann, and John M Wassick (2020). "OR-Gym: A Reinforcement Learning Library for Operations Research Problem". In: *arXiv preprint arXiv:2008.06319* (cit. on p. 11).

Hurink, Johann, Bernd Jurisch, and Monika Thole (1994). "Tabu search for the job-shop scheduling problem with multi-purpose machines". In: *Operations-Research-Spektrum* 15.4, pp. 205–215 (cit. on p. 12).

Hussain, Azham, Emmanuel OC Mkpojiogu, and Fazillah Mohmad Kamal (2016). "The role of requirements in the success or failure of software projects". In: *International Review of Management and Marketing* 6.7, pp. 306–311 (cit. on p. 54).

IEEE Computer Society. Software Engineering Technical Committee (1983). *IEEE Standard Glossary of Software Engineering Terminology: An American National Standard*. IEEE (cit. on p. 54).

Jan't Hoen, Pieter, Karl Tuyls, Liviu Panait, Sean Luke, and Johannes A La Poutre (2005). "An overview of cooperative and competitive multiagent learning". In: *International Workshop on Learning and Adaption in Multi-Agent Systems*. Springer, pp. 1–46 (cit. on p. 41).

Jiménez, Yailen Martínez (2012). "A generic multi-agent reinforcement learning approach for scheduling problems". In: *PhD, Vrije Universiteit Brussel*, p. 128 (cit. on pp. 14, 19, 20, 29, 30, 34, 36, 38, 41, 44, 59, 141, VI, IX, XIII).

Jones, Albert, Luis C Rabelo, and Abeer T Sharawi (1998). "Survey of job shop scheduling techniques". In: *NISTIR, National Institute of Standards and Technology, Gaithersburg, MD* (cit. on p. 12).

Karlsson, Joachim (1996). "Software requirements prioritizing". In: *Proceedings of the Second International Conference on Requirements Engineering*. IEEE, pp. 110–116 (cit. on p. 54).

Kayhan, Behice Meltem and Gokalp Yildiz (2021). "Reinforcement Learning Applications To Machine Scheduling Problems: a Comprehensive Literature Review". In: *Journal of Intelligent Manufacturing*. DOI: 10.1007/s10845-021-01847-3 (cit. on pp. 12–14).

Kendall, Maurice G (1938). "A new measure of rank correlation". In: *Biometrika* 30.1/2, pp. 81–93 (cit. on p. 137).

Kim, GH and GSG Lee (1998). "Genetic Reinforcement Learning Approach To the Heterogeneous Machine Scheduling Problem". In: *IEEE Transactions on Robotics and Automation* 14.6, pp. 879–893 (cit. on pp. 9, 29).

Kim, Hyun-Jung and Jun-Ho Lee (2022). "Scheduling of Dual-Gripper Robotic Cells with Reinforcement Learning". In: *IEEE Transactions on Automation Science and Engineering* 19.2, pp. 1120–1136. DOI: 10.1109/TASE.2020.3047924 (cit. on pp. 20, 40, V, IX, XII).

Kipf, Thomas N and Max Welling (2016). "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (cit. on p. 41).

Knust, Sigrid (2000). "Shop-scheduling problems with transportation". In: *Osnabrük University Library* (cit. on p. 11).

Kocsis, Levente and Csaba Szepesvári (2006). "Bandit based monte-carlo planning". In: *European conference on machine learning*. Springer, pp. 282–293 (cit. on p. 117).

Kohonen, Teuvo (1990). "The self-organizing map". In: *Proceedings of the IEEE* 78.9, pp. 1464–1480 (cit. on p. 41).

Kuball, Michael (2022). *Quo Vadis MES? Eine Wettbewerbsanalyse von Manufacturing Execution Systems hinsichtlich der Eignung zur Produktionssteuerung* (cit. on p. 106).

Kuhn, Max, Kjell Johnson, et al. (2013). *Applied predictive modeling*. Vol. 26. Springer (cit. on p. 143).

Kuhnle, Alexander, Michael Schaarschmidt, and Kai Fricke (2017). *Tensorforce: a TensorFlow library for applied reinforcement learning*. Web page. URL: https://github.com/tensorforce/tensorforce (cit. on p. 11).

Kühnle, Alexander, Michael Schaarschmidt, and Kai Fricke (2017). "Tensorforce: a TensorFlow library for applied reinforcement learning". In: *GitHub repository*. https://github.com/tensorforce/tensorforce (cit. on p. 53).

Kuhnle, Andreas, Jan-Philipp Kaiser, Felix Theiß, Nicole Stricker, and Gisela Lanza (2020). "Designing an adaptive production control system using reinforcement learning". In: *Journal of Intelligent Manufacturing*, pp. 1–22 (cit. on pp. 20, 22, 23, 25, 29, 36, 43, 44, 46, 47, 49, 51, 59, 61, 69, 88, 131, 133, 135, 136, VII, X, XIII).

Kuhnle, Andreas, Louis Schaefer, Nicole Stricker, and Gisela Lanza (2019). "Design, Implementation and Evaluation of Reinforcement Learning for an Adaptive Order Dispatching in Job Shop Manufacturing Systems". In: *52nd CIRP Conference on Manufacturing Systems (CMS)*. Ed. by P Butala, E Govekar, and R Vrabic. Vol. 81. Procedia CIRP, pp. 234–239. DOI: 10.1016/j.procir.2019.03.041 (cit. on pp. 11, 14, 25, VI, X, XIII).

Lang, Sebastian, Fabian Behrendt, Nico Lanzerath, Tobias Reggelin, and Marcel Mueller (2020). "Integration of Deep Reinforcement Learning and Discrete-Event Simulation for Real-Time Scheduling of a Flexible Job Shop Production". In: *2020 Winter Simulation Conference (WSC)*. Winter Simulation Conference Proceedings, pp. 3057–3068. DOI: 10.1109/WSC48552.2020.9383997 (cit. on pp. 38, IV, VIII, XII).

Lawler, Eugene L, Jan Karel Lenstra, Alexander HG Rinnooy Kan, and David B Shmoys (1993). "Sequencing and scheduling: Algorithms and complexity". In: *Handbooks in operations research and management science* 4, pp. 445–522 (cit. on p. 11).

Lawrence, S (1984). "Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement)". In: *Graduate School of Industrial Administration, Carnegie-Mellon University* (cit. on p. 12).

Lee, Hochang and Dong-Won Seo (2016). "Performance evaluation of WIP-controlled line production systems with constant processing times". In: *Computers & Industrial Engineering* 94, pp. 138–146 (cit. on p. 126).

Lee, Jun-Ho and Hyun-Jung Kim (2022). "Reinforcement Learning for Robotic Flow Shop Scheduling with Processing Time Variations". In: *International Journal of Production Research* 60.7, pp. 2346–2368. DOI: 10.1080/00207543.2021.1887533 (cit. on pp. 18, VII, X, XIV).

Li, Jing, Xingye Dong, Kai Zhang, and Sheng Han (2020). "Solving Open Shop Scheduling Problem via Graph Attention Neural Network". In: *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. Ed. by M Alamaniotis and S Pan. Proceedings-International Conference on Tools With Artificial Intelligence, pp. 277–284. DOI: 10.1109/ICTAI50040.2020.00052 (cit. on pp. 31, 39, IV, VIII, XII).

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2015). "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (cit. on pp. 39, 97).

Lin, Chengran, Zhengcai Cao, and Mengchu Zhou (2022). "Learning-Based Grey Wolf Optimizer for Stochastic Flexible Job Shop Scheduling". In: *IEEE Transactions on Automation Science and Engineering*. DOI: 10.1109/TASE.2021.3129439 (cit. on pp. 22, VII, X, XIV).

Lin, Chun-Cheng, Der-Jiunn Deng, Yen-Ling Chih, and Hsin-Ting Chiu (2019). "Smart Manufacturing Scheduling with Edge Computing Using Multiclass Deep Q Network". In: *IEEE Transactions on Industrial Informatics* 15.7, pp. 4276–4284. DOI: 10.1109/TII.2019.2908210 (cit. on pp. 36, IV, VIII, XI).

Lin, Wang Guolei Zhong Shisheng Lin (2009). "Clustering state membership-based Q-learning for dynamic scheduling". In: *Chinese High Technology Letters* 4 (cit. on p. 43).

Lindauer, Marius, Holger H Hoos, Frank Hutter, and Torsten Schaub (2015). "Autofolio: An automatically configured algorithm selector". In: *Journal of Artificial Intelligence Research* 53, pp. 745–778 (cit. on p. 151).

Liu, Chien-Liang, Chuan-Chin Chang, and Chun-Jan Tseng (2020). "Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems". In: *IEEE Access* 8, pp. 71752–71762 (cit. on pp. 51, 135, VI, X, XIII).

Long, Xiaojun, Jingtao Zhang, Xing Qi, Wenlong Xu, Tianguo Jin, and Kai Zhou (2022). "A Self-Learning Artificial Bee Colony Algorithm Based on Reinforcement Learning for a Flexible Job-Shop Scheduling Problem". In: *Concurrency and Computation-Practice&Experience* 34.4. DOI: 10.1002/cpe.6658 (cit. on pp. V, IX, XII).

Luo, Shu (2020). "Dynamic Scheduling for Flexible Job Shop with New Job Insertions by Deep Reinforcement Learning". In: *Applied Soft Computing* 91. DOI: 10.1016/j.asoc.2020.106208 (cit. on pp. 20, 29, 35, 43, 51, 60, 61, 83, 88, 133, 135, VII, X, XIII).

Luo, Shu, Linxuan Zhang, and Yushun Fan (2021a). "Dynamic Multi-Objective Scheduling for Flexible Job Shop by Deep Reinforcement Learning". In: *Computers&Industrial Engineering* 159. DOI: 10.1016/j.cie.2021.107489 (cit. on pp. 49, 127, VII, X, XIII).

– (2021b). "Real-Time Scheduling for Dynamic Partial-No-wait Multiobjective Flexible Job Shop by Deep Reinforcement Learning". In: *IEEE Transactions on Automation Science and Engineering*. DOI: 10.1109/TASE.2021.3104716 (cit. on pp. 127, VII, X, XIV).

Macal, Charles M and Michael J North (2005). "Tutorial on agent-based modeling and simulation". In: *Proceedings of the Winter Simulation Conference, 2005.* IEEE, 14–pp (cit. on p. 42).

Mahadevan, Sridhar and Georgios Theocharous (1998). "Optimizing Production Manufacturing Using Reinforcement Learning." In: *FLAIRS Conference*. Vol. 372, p. 377 (cit. on pp. 22, 25, 29, V, IX, XII).

Marandi, Fateme and S. M. T. Fatemi Ghomi (2019). "Network Configuration Multi-Factory Scheduling with Batch Delivery: a Learning-Oriented Simulated Annealing Approach". In: *Computers&Industrial Engineering* 132, pp. 293–310. DOI: 10.1016/j.cie.2019.04.032 (cit. on pp. IV, VIII, XI).

Martinez, EC (1999). "Solving Batch Process Scheduling/planning Tasks Using Reinforcement Learning". In: *Computers&Chemical Engineering* 23.S, S527–S530. DOI: 10.1016/S0098-1354(99)80130-6 (cit. on pp. III, VIII, XI).

Martinez Jimenez, Yailen, Jessica Coto Palacio, and Ann Nowe (2020). "Multi-Agent Reinforcement Learning Tool for Job Shop Scheduling Problems". In: *Optimization and Learning*. Ed. by B Dorronsoro, P Ruiz, JC DeLaTorre, D Urda, and EG Talbi. Vol. 1173. Communications in Computer and Information Science, pp. 3–12. DOI: 10.1007/978-3-030-41913-4_1 (cit. on pp. IV, VIII, XII).

Martins, Miguel S. E., Joaquim L. Viegas, Tiago Coito, Bernardo Marreiros Firme, Joao M. C. Sousa, Joao Figueiredo, and Susana M. Vieira (2020). "Reinforcement Learning for Dual-Resource Constrained Scheduling". In: *IFAC Papersonline* 53.2, pp. 10810–10815. DOI: 10.1016/j.ifacol.2020.12.2866 (cit. on pp. 21, IV, VIII, XII).

Martínez, Yailen, Ann Nowé, Juliett Suárez, and Rafael Bello (2011). "A reinforcement learning approach for the flexible job shop scheduling problem". In: *International Conference on Learning and Intelligent Optimization*. Springer, pp. 253–262 (cit. on pp. 20, 29, 41, IV, VIII, XI).

Mastrolli, Monaldo (1998). *FJSPLIB*. Web Page. URL: https://people.idsia.ch/~monaldo/fjsp.html (cit. on pp. 12, 17, 62, 69, 124, 125).

Matignon, Laëtitia, Guillaume J Laurent, and Nadine Le Fort-Piat (2006). "Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning". In: *International Conference on Artificial Neural Networks*. Springer, pp. 840–849 (cit. on p. 38).

Mendez-Hernandez, Beatriz M., Erick D. Rodriguez-Bazan, Yailen Martinez-Jimenez, Pieter Libin, and Ann Nowe (2019). "A Multi-Objective Reinforcement Learning Algorithm for JSSP". In: *Artificial Neural Networks and Machine Learning - ICANN 2019: Theoretical Neural Computation, Pt I*. Ed. by IV Tetko, V Kurkova, P Karpov, and F Theis. Vol. 11727. Lecture

Notes in Computer Science, pp. 567–584. DOI: 10.1007/978-3-030-30487-4_44 (cit. on pp. 25, 51, 141, IV, VIII, XI).

MES-D.A.CH-Verband (2021). *Marktspiegel MES*. URL: https://www.checkvision.de/upload/pdfs/896.pdf (cit. on p. 106).

Min, Byungwook and Chang Ouk Kim (2022). "State-Dependent Parameter Tuning of the Apparent Tardiness Cost Dispatching Rule Using Deep Reinforcement Learning". In: *IEEE Access* 10, pp. 20187–20198. DOI: 10.1109/ACCESS.2022.3152192 (cit. on pp. 31, 37, VII, XI, XIV).

Mladenović, Nenad and Pierre Hansen (1997). "Variable neighborhood search". In: *Computers &operations research* 24.11, pp. 1097–1100 (cit. on p. 31).

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 1928–1937 (cit. on p. 39).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (cit. on p. 39).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533 (cit. on pp. 52, 96).

Mohan, Jatoth, Krishnanand Lanka, and A Neelakanteswara Rao (2019). "A review of dynamic job shop scheduling techniques". In: *Procedia Manufacturing* 30, pp. 34–39 (cit. on p. 12).

Montillet, Jean-Philippe, Kegen Yu, Lukasz Kosma Bonenberg, and Gethin Wyn Roberts (2016). "Optimization algorithms in local and global positioning". In: *Handbook of Research on Modern Optimization Algorithms and Applications in Engineering and Economics*. IGI Global, pp. 1–53 (cit. on p. 87).

Moon, Junhyung, Minyeol Yang, and Jongpil Jeong (2021). "A Novel Approach To the Job Shop Scheduling Problem Based on the Deep Q-Network in a Cooperative Multi-Access Edge Computing Ecosystem". In: *Sensors* 21.13. DOI: 10.3390/s21134553 (cit. on pp. V, IX, XII).

Morse, Janice M (2010). *"Cherry picking": Writing from thin data* (cit. on p. 4).

Moser, Joshua, Julia Hoffman, Robert Hildebrand, and Erik Komendera (2020). "A Flexible Job Shop Scheduling Representation of the Autonomous In-Space Assembly Task Assignment Problem". In: *arXiv preprint arXiv:2003.12148* (cit. on pp. 19, IV, VIII, XII).

Müller, Klaus G., Tony Vignaux, Ontje Lünsdorf, Stefan Scherfke, Karen Turner, Johannes Koomer, Steven Kennedy, Matthew Grogan, Sean Reed, Christoph Körner, Andreas Beham, Larissa Reis, Peter Grayson, and Cristian Klein (2020). *SimPy*. URL: https://pypi.org/project/simpy/ (cit. on p. 63).

Newell, David (2022). *dagre - Graph layout for JavaScript*. URL: https://github.com/dagrejs/dagre (cit. on p. 80).

Ni, Fei, Jianye Hao, Jiawen Lu, Xialiang Tong, Mingxuan Yuan, Jiahui Duan, Yi Ma, and Kun He (2021). "A Multi-Graph Attributed Reinforcement Learning Based Optimization Algorithm for Large-Scale Hybrid Flow Shop Scheduling Problem". In: *KDD ' 21: Proceedings of the 27th ACM Sigkdd Conference on Knowledge Discovery&Data Mining*, pp. 3441–3451. DOI: 10.1145/3447548.3467135 (cit. on pp. V, IX, XII).

Oh, Seog-Chan, James W Wells, and Jorge Arinez (2022). "Conveyor-Less Urban-Car Assembly Factory with VaaC and Matrix System". In: *Smart Cities* 5.3, pp. 947–963 (cit. on p. 17).

Palombarini, Jorge and Ernesto Martinez (2012). "Smartgantt - an Intelligent System for Real Time Rescheduling Based on Relational Reinforcement Learning". In: *Expert Systems with Applications* 39.11, pp. 10251–10268. DOI: 10.1016/j.eswa.2012.02.176 (cit. on pp. VI, X, XIII).

Palombarini, Jorge A. and Ernesto C. Martinez (2019). "Closed-Loop Rescheduling Using Deep Reinforcement Learning". In: *IFAC Papersonline* 52.1, pp. 231–236. DOI: 10.1016/j.ifacol.2019.06.067 (cit. on pp. 30, VI, X, XIII).

Pan, Zixiao, Ling Wang, Jingjing Wang, and Jiawen Lu (2021). "Deep Reinforcement Learning Based Optimization Algorithm for Permutation Flow-Shop Scheduling". In: *IEEE Transactions on Emerging Topics in Computational Intelligence*. DOI: 10.1109/TETCI.2021.3098354 (cit. on pp. 31, 39, V, IX, XII).

Panwalkar, Shrikant S and Wafik Iskander (1977). "A survey of scheduling rules". In: *Operations research* 25.1, pp. 45–61 (cit. on p. 107).

Park, Huiung, Haeyong Kim, Seon-Tae Kim, and Pyeongsoo Mah (2020a). "Multi-Agent Reinforcement-Learning-based Time-Slotted Channel Hopping Medium Access Control Scheduling Scheme". In: *IEEE Access* 8, pp. 139727–139736. DOI: 10.1109/ACCESS.2020.3010575 (cit. on p. 9).

Park, In-Beom, Jaeseok Huh, Joongkyun Kim, and Jonghun Park (2019). "A Reinforcement Learning Approach to Robust Scheduling of Semiconductor Manufacturing Facilities". In: *IEEE Transactions on Automation Science and Engineering* (cit. on pp. 135, 145, VI, X, XIII).

– (2020b). "A Reinforcement Learning Approach To Robust Scheduling of Semiconductor Manufacturing Facilities". In: *IEEE Transactions on Automation Science and Engineering* 17.3, pp. 1420–1431. DOI: 10.1109/TASE.2019.2956762 (cit. on pp. 19, VII, X, XIII).

Park, Junyoung, Jaehyeong Chun, Sang Hun Kim, Youngkook Kim, and Jinkyoo Park (2021). "Learning To Schedule Job-Shop Problems: Representation and Policy Learning Using Graph Neural Network and Reinforcement Learning". In: *International Journal of Production Research* 59.11, pp. 3360–3377. DOI: 10.1080/00207543.2020.1870013 (cit. on pp. 48, 141, V, IX, XII).

Paternina-Arboleda, Carlos D and Tapas K Das (2005). "A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem". In: *Simulation Modelling Practice and Theory* 13.5, pp. 389–406 (cit. on pp. 22, 25, 35, 41, 44, V, IX, XIII).

Perron, Laurent and Vincent Furnon (Aug. 18, 2022). *Or-Tools*. Version v9.6. Google. URL: \url{https://developers.google.com/optimization} (cit. on pp. 63, 113).

Petrova, Irina, Arina Buzdalova, and Maxim Buzdalov (2013). “Improved Helper-Objective Optimization Strategy for Job-Shop Scheduling Problem”. In: *2013 12th International Conference on Machine Learning and Applications (ICMLA 2013), Vol 2*. Ed. by MA Wani, G Tecuci, M Boicu, M Kubat, TM Khoshgoftaar, and N Seliya, pp. 374–377. DOI: 10.1109/ICMLA.2013.151 (cit. on pp. IV, VIII, XI).

Pilchau, Wenzel Baron Pilar von, Anthony Stein, and Jörg Hähner (2020). “Bootstrapping a DQN Replay Memory with Synthetic Experiences”. In: *CoRR* abs/2002.01370. arXiv: 2002.01370. URL: https://arxiv.org/abs/2002.01370 (cit. on p. 96).

Pinedo, Michael (2012). *Scheduling*. Vol. 29. Springer (cit. on pp. 9, 11, 13–19, 22–24, 56, 110).

Plappert, Matthias (2016). “keras-rl”. In: *GitHub repository*. https://github.com/keras-rl/keras-rl (cit. on p. 53).

Pol, Sebastian, Schirin Baer, Danielle Turner, Vladimir Samsonov, and Tobias Meisen (2021). “Global Reward Design for Cooperative Agents To Achieve Flexible Production Control under Real-Time Constraints”. In: *Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS 2021), Vol 1*. Ed. by J Filipe, M Smialek, A Brodsky, and S Hammoudi, pp. 515–526. DOI: 10.5220/0010455805150526 (cit. on pp. 42, V, IX, XII).

Qu, Shuhui, Tianshu Chu, Jie Wang, James Leckie, and Weiwen Jian (2015). “A centralized reinforcement learning approach for proactive scheduling in manufacturing”. In: *2015 IEEE 20th Conference on Emerging Technologies &Factory Automation (ETFA)*. IEEE, pp. 1–8 (cit. on pp. 22, 24, 25, 29, 35, 42, 44, 59, 61, VI, X, XIII).

Qu, Shuhui, Jie Wang, and Govil Shivani (2016). “Learning Adaptive Dispatching Rules for a Manufacturing Process System by Using Reinforcement Learning Approach”. In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE International Conference on Emerging Technologies and Factory Automation-ETFA (cit. on pp. VI, X, XIII).

Racanière, Sébastien, Théophane Weber, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. (2017). “Imagination-Augmented Agents for Deep Reinforcement Learning.” In: *NIPS*, pp. 5690–5701 (cit. on p. 40).

Raffin, Antonin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann (2021). “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268, pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html (cit. on pp. 53, 96, 97).

Ren, J. F., C. M. Ye, and Y. Li (2021a). “A New Solution To Distributed Permutation Flow Shop Scheduling Problem Based on NASH Q-Learning”. In: *Advances in Production Engineering&Management* 16.3, pp. 269–284. DOI: 10.14743/apem2021.3.399 (cit. on pp. V, IX, XII).

Ren, J. F., C. M. Ye, and F. Yang (2020). “A Novel Solution To JSPS Based on Long Short-Term Memory and Policy Gradient Algorithm”. In: *International Journal of Simulation Modelling* 19.1, pp. 157–168. DOI: 10.2507/IJSIMM19-1-CO4 (cit. on pp. IV, VIII, XII).

Ren, Jianfeng, Chunming Ye, and Feng Yang (2021b). “Solving Flow-Shop Scheduling Problem with a Reinforcement Learning Algorithm that Generalizes the Value Function

with Neural Network". In: *Alexandria Engineering Journal* 60.3, pp. 2787–2800. DOI: 10.1016/j.aej.2021.01.030 (cit. on pp. 48, V, IX, XII).

Reyes, John, Darwin Aldas, Kevin Alvarez, Mario García, and Mery Ruíz (2017). "The Factory Physics for the Scheduling: Application to Footwear Industry." In: *SIMULTECH*, pp. 248–254 (cit. on p. 25).

Reyna, Yunior César Fonseca, Yailen Martínez Jiménez, Juan Manuel Bermúdez Cabrera, and Beatriz M Méndez Hernández (2015). "A reinforcement learning approach for scheduling problems". In: *Investigación Operacional* 36.3, pp. 225–231 (cit. on pp. 14, 44, 51, 141, IV, VIII, XI).

Riedmiller, S and M Riedmiller (1999). "A Neural Reinforcement Learning Approach To Learn Local Dispatching Policies in Production Scheduling". In: *Ijcai-99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Vols 1&2*, pp. 764–769 (cit. on pp. 29, 34, 135, III, VIII, XI).

Rinciog, Alexandru and Anne Meyer (2021a). "Fabricatio-rl: a reinforcement learning simulation framework for production scheduling". In: *2021 Winter Simulation Conference (WSC)*. IEEE, pp. 1–12. DOI: 10.1109/WSC52266.2021.9715366 (cit. on pp. 6, 7, 80).

– (2021b). *FabricatioRL*. https://github.com/malerinc/fabricatio-rl.git (cit. on p. 6).

– (2021c). "Towards Standardizing Reinforcement Learning Approaches for Stochastic Production Scheduling". In: *CoRR* abs/2104.08196. arXiv: 2104.08196 (cit. on p. 6).

– (2022). "Towards Standardising Reinforcement Learning Approaches for Production Scheduling Problems". In: *Procedia CIRP* 107, pp. 1112–1119. DOI: 10.1016/j.procir.2022.05.117 (cit. on pp. 6, 7).

Rinciog, Alexandru, Carina Mieth, Paul Maria Scheikl, and Anne Meyer (2020). "Sheet-Metal Production Scheduling Using AlphaGo Zero". In: *Proceedings of the Conference on Production Systems and Logistics: CPSL 2020*. Hannover: Institutionelles Repositorium der Leibniz Universität Hannover, pp. 342–352. DOI: 10.15488/9640 (cit. on pp. 7, 19, 22, 25, 29, 34, 41, 51, 88, 99, 141, 142, IV, VIII, XI).

Robertson, Suzanne and James Robertson (2012). *Mastering the requirements process: Getting requirements right*. Addison-wesley (cit. on p. 54).

Ruiz, Rubén and José Antonio Vázquez-Rodriguez (2010). "The hybrid flow shop scheduling problem". In: *European journal of operational research* 205.1, pp. 1–18 (cit. on p. 14).

Rummery, Gavin A and Mahesan Niranjan (1994). *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK (cit. on p. 39).

Samsonov, Vladimir, Marco Kemmerling, Maren Paegert, Daniel Luetticke, Frederick Sauermann, Andreas Guetzlaff, Gunther Schuh, and Tobias Meisen (2021). "Manufacturing Control in Job Shop Environments with Reinforcement Learning". In: *ICAART: Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Vol 2*. Ed. by AP Rocha, L Steels, and J VandenHerik, pp. 589–597. DOI: 10.5220/0010202405890597 (cit. on pp. 37, 48, V, IX, XII).

Satopaa, Ville, Jeannie Albrecht, David Irwin, and Barath Raghavan (2011). "Finding a" kneedle" in a haystack: Detecting knee points in system behavior". In: *2011 31st*

international conference on distributed computing systems workshops. IEEE, pp. 166–171 (cit. on p. 127).

Schrittwieser, Julian, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. (2020). "Mastering atari, go, chess and shogi by planning with a learned model". In: *Nature* 588.7839, pp. 604–609 (cit. on p. 100).

Schuh, Günther, Volker Stich, and H Wienholdt (2013). *Logistikmanagement*. Springer (cit. on p. 23).

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (2015). "Trust region policy optimization". In: *International conference on machine learning*, pp. 1889–1897 (cit. on p. 39).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (cit. on p. 39).

Sculley, David, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison (2015). "Hidden technical debt in machine learning systems". In: *Advances in neural information processing systems*, pp. 2503–2511 (cit. on p. 4).

Seito, Takanari and Satoshi Munakata (2020). "Production Scheduling Based on Deep Reinforcement Learning Using Graph Convolutional Neural Network". In: *Icaart: Proceedings of the 12th International Conference on Agents and Artificial Intelligence, Vol 2*. Ed. by AP Rocha, L Steels, and J VanDenHerik, pp. 766–772. DOI: 10.5220/0009095207660772 (cit. on pp. 31, V, IX, XII).

Serrano-Ruiz, Julio C., Josefa Mula, and Raul Poler (2022). "Development of a Multidimensional Conceptual Model for Job Shop Smart Manufacturing Scheduling from the Industry 4.0 Perspective". In: *Journal of Manufacturing Systems* 63, pp. 185–202. DOI: 10.1016/j.jmsy.2022.03.011 (cit. on p. 9).

Serrano Ruiz, Julio Cesar, Josefa Mula Bru, and Raul Poler Escoto (2021). "Smart Digital Twin for Zdm-Based Job-Shop Scheduling". In: *2021 IEEE International Workshop on Metrology for Industry 4.0&IOT (IEEE Metroind4.0&IOT)*, pp. 510–515. DOI: 10.1109/METROIND4.0IOT51437.2021.9488473 (cit. on p. 9).

Shahrabi, Jamal, Mohammad Amin Adibi, and Masoud Mahootchi (2017). "A Reinforcement Learning Approach To Parameter Estimation in Dynamic Job Shop Scheduling". In: *Computers&Industrial Engineering* 110, pp. 75–82. DOI: 10.1016/j.cie.2017.05.026 (cit. on pp. 31, 35, 37, 43, 49, 60, VI, X, XIII).

Shiue, Yeou-Ren, Ken-Chuan Lee, and Chao-Ton Su (2018). "Real-Time Scheduling for a Smart Factory Using a Reinforcement Learning Approach". In: *Computers&Industrial Engineering* 125, pp. 604–614. DOI: 10.1016/j.cie.2018.03.039 (cit. on pp. 40, VI, X, XIII).

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, p. 484 (cit. on pp. 27, 52, 100).

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2017a). "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv preprint arXiv:1712.01815* (cit. on pp. 27, 100).

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. (2017b). "Mastering the game of go without human knowledge". In: *nature* 550.7676, pp. 354–359 (cit. on pp. 39, 61, 100, 102, 103).

Slotnick, Susan A (2011). "Order acceptance and scheduling: A taxonomy and review". In: *European Journal of Operational Research* 212.1, pp. 1–11 (cit. on pp. 12, 13).

Storer, Robert H, S David Wu, and Renzo Vaccari (1992). "New search spaces for sequencing problems with application to job shop scheduling". In: *Management science* 38.10, pp. 1495–1509 (cit. on p. 12).

Stricker, Nicole, Andreas Kuhnle, Roland Sturm, and Simon Friess (2018). "Reinforcement learning for adaptive order dispatching in the semiconductor industry". In: *CIRP Annals* 67.1, pp. 511–514 (cit. on pp. 19, 43, 44, 145, VI, X, XIII).

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press (cit. on pp. 26, 38–40, 61, 89, 90, 92–94).

Sutton, Richard S, Andrew G Barto, et al. (1998). *Introduction to reinforcement learning*. Vol. 2. 4. MIT press Cambridge (cit. on pp. 89, 97).

Taillard, Eric (1993). "Benchmarks for basic scheduling problems". In: *european journal of operational research* 64.2, pp. 278–285 (cit. on p. 12).

Tan, Qingmeng, Yifei Tong, Shaofeng Wu, and Dongbo Li (2019). "Modeling, Planning, and Scheduling of Shop-Floor Assembly Process with Dynamic Cyber-Physical Interactions: a Case Study for Cps-Based Smart Industrial Robot Production". In: *International Journal of Advanced Manufacturing Technology* 105.9, SI, pp. 3979–3989. DOI: 10.1007/s00170-019-03940-7 (cit. on pp. IV, VIII, XI).

Thomas, Tara Elizabeth, Jinkyu Koo, Somali Chaterji, and Saurabh Bagchi (2018). "Minerva: a Reinforcement Learning-Based Technique for Optimal Scheduling and Bottleneck Detection in Distributed Factory Operations". In: *2018 10th International Conference on Communication Systems&Networks (Comsnets)*. International Conference on Communication Systems and Networks, pp. 129–136 (cit. on pp. 19, 20, 23, 25, 29, 35, 36, 60, 135, 141, VI, X, XIII).

Thulasiraman, Krishnaiyan and Madisetti NS Swamy (2011). *Graphs: theory and algorithms*. John Wiley & Sons (cit. on p. 17).

Ugarte, B Saenz de, A Artiba, and R Pellerin (2009). "Manufacturing execution system–a literature review". In: *Production planning and control* 20.6, pp. 525–539 (cit. on p. 106).

Van Hasselt, Hado, Arthur Guez, and David Silver (2016). "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30 (cit. on pp. 39, 96).

Villalón, CC, T Stützle, and M Dorigo (2021). "Cuckoo search≡(μ+ λ)–evolution strategy". In: *IRIDIA–Technical Report Series* (cit. on p. 31).

Waibel, Alexander, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang (1989). "Phoneme recognition using time-delay neural networks". In: *IEEE transactions on acoustics, speech, and signal processing* 37.3, pp. 328–339 (cit. on p. 41).

Wang, Hao-Xiang and Hong-Sen Yan (2016). "An interoperable adaptive scheduling strategy for knowledgeable manufacturing based on SMGWQ-learning". In: *Journal of Intelligent Manufacturing* 27.5, pp. 1085–1095 (cit. on p. 43).

Wang, Haoxiang, Bhaba R. Sarker, Jing Li, and Jian Li (2021a). "Adaptive Scheduling for Assembly Job Shop with Uncertain Assembly Times Based on Dual Q-Learning". In: *International Journal of Production Research* 59.19, pp. 5867–5883. doi: 10.1080/00207543.2020.1794075 (cit. on pp. 42, VII, X, XIII).

Wang, Jing-Jing and Ling Wang (2022). "A Cooperative Memetic Algorithm with Learning-Based Agent for Energy-Aware Distributed Hybrid Flow-Shop Scheduling". In: *IEEE Transactions on Evolutionary Computation* 26.3, pp. 461–475. doi: 10.1109/TEVC.2021.3106168 (cit. on pp. 25, V, IX, XII).

Wang, Libing, Xin Hu, Yin Wang, Sujie Xu, Shijun Ma, Kexin Yang, Zhijun Liu, and Weidong Wang (2021b). "Dynamic Job-Shop Scheduling in Smart Manufacturing Using Deep Reinforcement Learning". In: *Computer Networks* 190. doi: 10.1016/j.comnet.2021.107969 (cit. on pp. 48, V, IX, XII).

Wang, Ling, Zixiao Pan, and Jingjing Wang (2021c). "A review of reinforcement learning based intelligent optimization for manufacturing scheduling". In: *Complex System Modeling and Simulation* 1.4, pp. 257–270 (cit. on p. 88).

Wang, Yanting, Zhengwen He, Louis-Phillipe Kerkhove, and Mario Vanhoucke (2017). "On the performance of priority rules for the stochastic resource constrained multi-project scheduling problem". In: *Computers & industrial engineering* 114, pp. 223–234 (cit. on p. 88).

Wang, Yi-Chi and John M Usher (2005). "Application of reinforcement learning for agent-based production scheduling". In: *Engineering Applications of Artificial Intelligence* 18.1, pp. 73–82 (cit. on pp. 24, 35, 60, 135, V, IX, XII).

Wang, Yi-Chi and John M. Usher (2007). "A Reinforcement Learning Approach for Developing Routing Policies in Multi-Agent Production Scheduling". In: *International Journal of Advanced Manufacturing Technology* 33.3-4, pp. 323–333. doi: 10.1007/s00170-006-0465-y (cit. on pp. 19, 127, V, IX, XIII).

Wang, Yu-Fang (2020). "Adaptive job shop scheduling strategy based on weighted Q-learning algorithm". In: *Journal of Intelligent Manufacturing* 31.2, pp. 417–432 (cit. on pp. 24, 25, 35, 43, 61, 135, VII, X, XIII).

Wang, Ziyu, Nando de Freitas, and Marc Lanctot (2015). "Dueling Network Architectures for Deep Reinforcement Learning". In: *CoRR* abs/1511.06581. arXiv: 1511.06581. url: http://arxiv.org/abs/1511.06581 (cit. on p. 39).

Warps, Time (1983). "String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison". In: *Addison-Wesley, Reading, MA). Strategies for studying heterogeneous genetic traits in humans by using a linkage map of restriction fragment length polymorphisms, Proc. Nut. Acad. Sei. US. A* 83, pp. 73–53 (cit. on p. 137).

Waschneck, Bernd, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, and Andreas Kyek (2018). "Optimization of global production scheduling with deep reinforcement learning". In: *Procedia CIRP* 72.1, pp. 1264–1269 (cit. on pp. 29, 34, 43, 87, 88, 127, 141, 145, VI, X, XIII).

Waskom, Michael L. (2021). "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60, p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021 (cit. on p. 80).

Watkins, CJCH (1989). "Learning form delayed rewards". In: *Ph. D. thesis, King's College, University of Cambridge* (cit. on p. 39).

Wei, YZ and MY Zhao (2004). "Composite Rules Selection Using Reinforcement Learning for Dynamic Job-Shop Scheduling". In: *2004 IEEE Conference on Robotics, Automation and Mechatronics, Vols 1 and 2*, pp. 1083–1088 (cit. on pp. V, IX, XII).

Wiendahl, HH, A Kluth, and R Kipp (2021). "Marktspiegel Business Software: MES–Fertigungssteuerung 2021/2022". In: *Trovarit, Aachen* (cit. on p. 106).

Williams, R (1987). "A class of gradient-estimation algorithms for reinforcement learning in neural networks". In: *Proceedings of the International Conference on Neural Networks*, pp. II–601 (cit. on p. 39).

Winands, Mark HM (2015). "Monte-Carlo tree search in board games". In: *Handbook of Digital Games and Entertainment Technologies*, pp. 1–30 (cit. on pp. 117, 118).

Wu, Chen-Xin, Min-Hui Liao, Mumtaz Karatas, Sheng-Yong Chen, and Yu-Jun Zheng (2020). "Real-Time Neural Network Scheduling of Emergency Medical Mask Production during Covid-19". In: *Applied Soft Computing* 97.A. DOI: 10.1016/j.asoc.2020.106790 (cit. on pp. 18, 19, 31, VII, X, XIII).

Xue, Tianfang, Peng Zeng, and Haibin Yu (2018). "A Reinforcement Learning Method for Multi-Agv Scheduling in Manufacturing". In: *2018 IEEE International Conference on Industrial Technology (ICIT)*. IEEE International Conference on Industrial Technology, pp. 1557–1561. DOI: 10.1109/ICIT.2018.8352413 (cit. on pp. IV, VIII, XI).

Yamada, Takeshi and Ryohei Nakano (1992). "A genetic algorithm applicable to large-scale job-shop problems." In: *PPSN*. Vol. 2, pp. 281–290 (cit. on p. 12).

Yan, Chao-Bo, Lars Mönch, and Semyon M Meerkov (2019). "Characteristic curves and cycle time control of re-entrant lines". In: *IEEE Transactions on Semiconductor Manufacturing* 32.2, pp. 140–153 (cit. on pp. 126, 127).

Yang, H-B and H-S Yan (2009). "An adaptive approach to dynamic scheduling in knowledgeable manufacturing cell". In: *The International Journal of Advanced Manufacturing Technology* 42.3-4, p. 312 (cit. on p. 43).

Yang, RL, Velusamy Subramaniam, and Stanley B Gershwin (2006). "Setting real time WIP levels in production lines". In: *dspace.mit.edu* (cit. on p. 126).

Yang, Shengluo and Zhigang Xu (2021a). "Intelligent Scheduling and Reconfiguration via Deep Reinforcement Learning in Smart Manufacturing". In: *International Journal of Production Research*. DOI: 10.1080/00207543.2021.1943037 (cit. on pp. 18, 36, 42, VII, X, XIII).

Yang, Shengluo, Zhigang Xu, and Junyi Wang (2021b). “Intelligent Decision-Making of Scheduling for Dynamic Permutation Flowshop via Deep Reinforcement Learning”. In: *Sensors* 21.3. DOI: 10.3390/s21031019 (cit. on pp. 18, VII, X, XIII).

Yang, Wei-Ning and Barry L Nelson (1991). “Using common random numbers and control variates in multiple-comparison procedures”. In: *Operations Research* 39.4, pp. 583–591 (cit. on p. 62).

Yang, Xin-She and Suash Deb (2014). “Cuckoo search: recent advances and applications”. In: *Neural Computing and applications* 24, pp. 169–174 (cit. on p. 31).

Yingzi, Wei, Jiang Xinli, Hao Pingbo, and Gu Kanfeng (2009). “Pattern Driven Dynamic Scheduling Approach Using Reinforcement Learning”. In: *2009 IEEE International Conference on Automation and Logistics (ICAL 2009), Vols 1-3*, pp. 514+ (cit. on pp. VI, IX, XIII).

Yoshida, Takumi and Toshiyuki Kaneda (2009). “A Simulation Analysis of Shop-Around Behavior in a Commercial District as an Intelligent Agent Approach - a Case Study of OSU District of Nagoya City”. In: *Agent-Based Approaches in Economic and Social Complex Systems V*. Ed. by T Terano, H Kita, S Takahashi, and H Deguchi, pp. 131–142 (cit. on p. 9).

Yu, Tian, Jing Huang, and Qing Chang (2020). “Mastering the Working Sequence in Human-Robot Collaborative Assembly Based on Reinforcement Learning”. In: *IEEE Access* 8, pp. 163868–163877. DOI: 10.1109/ACCESS.2020.3021904 (cit. on pp. 99, IV, VIII, XII).

Yu, Xuefeng and Bala Ram (2006). “Bio-Inspired Scheduling for Dynamic Job Shops with Flexible Routing and Sequence-Dependent Setups”. In: *International Journal of Production Research* 44.22, pp. 4793–4813. DOI: 10.1080/00207540600621094 (cit. on p. 9).

Zahavy, Tom, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor (2018). “Learn what not to learn: Action elimination with deep reinforcement learning”. In: *Advances in neural information processing systems* 31 (cit. on p. 141).

Zaknich, Anthony (2005). *Principles of adaptive filters and self-learning systems*. Springer Science & Business Media (cit. on p. 87).

Zeng, Qingcheng, Zhongzhen Yang, and Xiangpei Hu (2011). “A Method Integrating Simulation and Reinforcement Learning for Operation Scheduling in Container Terminals”. In: *Transport* 26.4, pp. 383–393. DOI: 10.3846/16484142.2011.638022 (cit. on pp. 36, IV, VIII, XI).

Zhang, Cong, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi (2020). “Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning”. In: *Advances in Neural Information Processing Systems* 33 (cit. on pp. 31, 34, 51, 141, IV, VIII, XI).

Zhang, Puheng, Chuang Lin, Wenzhuo Li, and Xiao Ma (2017). “Long-Term Multi-Objective Task Scheduling with Diff-Serv in Hybrid Clouds”. In: *Web Information Systems Engineering, Wise 2017, Pt I*. Ed. by A Bouguettaya, Y Gao, A Klimenko, L Chen, X Zhang, F Dzerzhinskiy, W Jia, SV Klimenko, and Q Li. Vol. 10569. Lecture Notes in Computer Science, pp. 243–258. DOI: 10.1007/978-3-319-68783-4_17 (cit. on p. 9).

Zhang, Wei and Thomas G Dietterich (1995). "A reinforcement learning approach to job-shop scheduling". In: *IJCAI*. Vol. 95. Citeseer, pp. 1114–1120 (cit. on pp. 17, 19, 30, 41, III, VIII, XI).

– (1996). "High-performance job-shop scheduling with a time-delay TD (λ) network". In: *Advances in neural information processing systems*, pp. 1024–1030 (cit. on pp. 17, 19, 30, 37, 41, 51, III, VIII, XI).

Zhang, Zhicong, Weiping Wang, Shouyan Zhong, and Kaishun Hu (2013). "Flow Shop Scheduling with Reinforcement Learning". In: *Asia-Pacific Journal of Operational Research* 30.5. DOI: 10.1142/S0217595913500140 (cit. on pp. 41, 48, IV, VIII, XI).

Zhao, Fuqing, Lixin Zhang, Jie Cao, and Jianxin Tang (2021a). "A Cooperative Water Wave Optimization Algorithm with Reinforcement Learning for the Distributed Assembly No-Idle Flowshop Scheduling Problem". In: *Computers&Industrial Engineering* 153. DOI: 10.1016/j.cie.2020.107082 (cit. on pp. 31, V, IX, XII).

Zhao, Meng, Xinyu Li, Liang Gao, Ling Wang, and Mi Xiao (2019). "An improved Q-learning based rescheduling method for flexible job-shops with machine failures". In: *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. IEEE, pp. 331–337 (cit. on pp. 136, VI, X, XIII).

Zhao, Y. and H. Zhang (2021b). "Application of Machine Learning and Rule Scheduling in a Job-Shop Production Control System". In: *International Journal of Simulation Modelling* 20.2, pp. 410–421. DOI: 10.2507/IJSIMM20-2-CO10 (cit. on pp. VII, X, XIV).

Zhao, Yejian, Yanhong Wang, Yuanyuan Tan, Jun Zhang, and Hongxia Yu (2021c). "Dynamic Jobshop Scheduling Algorithm Based on Deep Q Network". In: *IEEE Access* 9, pp. 122995–123011. DOI: 10.1109/ACCESS.2021.3110242 (cit. on pp. 14, V, IX, XII).

Zhou, Longfei, Lin Zhang, and Berthold KP Horn (2020). "Deep reinforcement learning-based dynamic scheduling in smart manufacturing". In: *Procedia CIRP* 93, pp. 383–388 (cit. on pp. 25, 29, 35, 135, VI, X, XIII).

Zhu, Jialin, Huangang Wang, and Tao Zhang (2020). "A Deep Reinforcement Learning Approach To the Flexible Flowshop Scheduling Problem with Makespan Minimization". In: *Proceedings of 2020 IEEE 9th Data Driven Control and Learning Systems Conference (Ddcls'20)*. Ed. by M Sun and H Zhang, pp. 1220–1225 (cit. on pp. IV, IX, XII).

Appendices

Appendix A

Literature Overview Tables

The problem size in Table A.1 serves to give an orientation with regard to the difficulty of the combinatorial optimization problem. The following variables are used. The number of resources is specified using m, the number of jobs is marked by values of n, o_j indicates the maximum number of operations per job, m_{ave}^{alt} gives the average number of machine alternatives per operation, c indicates the number of work centers, o_{ave} denotes the average number of operations per job. For dynamic setups, we distinguish between the initial jobs known at time 0 and the total number of jobs by placing the latter values in parenthesis. Finally, in cases where vehicles are explicitly modeled ($tr(r)$, $tr(k,r)$) we use r to indicate their number.

Note that, depending on the setup different variables are needed fully define the size. In the case of Jm, for instance, the problem size is fully defined by the number of machines m and the number of jobs j. For FJc, the number of resources for every work center would additionally need to be specified. Owing to the orientative nature of the field and the potentially many variables needed to fully define complex setup sizes, we do not fully report the problem size. The reader is referred to the respective publications in such cases.

Table A.1: Scheduling Setups in Literature.

Source	Machine Setups (α)	Additional Constraints (β)	Optimization Goal (γ)	Instance Size Considered
		Deterministic Setups		
Zhang et al., 1995	$FPOc$	$recrc$	$C_{\max}$	n:3,m:82
Zhang et al., 1996	$FPOc$	$recrc$	$C_{\max}$	n:3,m:82
Riedmiller et al., 1999	Jm	$recrc$	$\sum T_j$	n:8, m:?
Martinez, 1999	FJc	$batch$, j_{batch}, $recrc$(?)	$\sum T_j$	n:?, m:14

Continued ...

TABLE A.1: SCHEDULING SETUPS IN LITERATURE. (Continued)

Source	Machine Setups (α)	Additional Constraints (β)	Optimization Goal (γ)	Instance Size Considered
Csaji et al., 2003	Jm	$tr(\infty)(?)$	$C_{\max}$	?
Hong et al., 2004	1	$fmls\|dbatch, recrc$	$f(T_j)\|f(Utl_i)$	n:100, m:1, o_j:2
Gabel et al., 2007a	Jm	None	$C_{\max}$	n:10, m:10
Gabel et al., 2007b	Jm	None	$C_{\max}$	n:20, m:15
Gabel, 2009	Jm	None	$C_{\max}$	n:20, m:15
Martínez et al., 2011	FJc	M_i^o	$C_{\max}$	n:20, m:15
Zeng et al., 2011	FFc	$s_{jk}, block_{in}$	$C_{\max}$	n:600, m:34, o_j:3
Petrova et al., 2013	Jm	None	$\sum F_j$	n:20,m:15
Zhang et al., 2013	Fm	None	$C_{\max}$	n:50, m:20
Reyna et al., 2015	Jm	None	$C_{\max}$	n:20, m:5
Arviv et al., 2016	FFc	$tr(r)$	$C_{\max}$	n:80, m:20
Fonseca-Reyna et al., 2018	Fm	None	$C_{\max}$	n:100, m:20
Xue et al., 2018	Fm	$tr(r)$	$C_{\max}$	n:50, m:50, r:2
Mendez-Hernandez et al., 2019	Fm	None	$C_{\max}\| \sum T_j$	n:20, m:15
Lin et al., 2019	Jm	None	$C_{\max}$	n:20, m:20
Tan et al., 2019	$Jm(?)$	None	$C_{\max}$	n:6, m:6
Marandi et al., 2019	$FPOc$	$tr(r)$	$f(T_j, Utl_{tr})$	n:50, m:20, o_j:30
Baer et al., 2019	FJc	?	$C_{\max}(?)$	?
Zhang et al., 2020	Jm	None	$C_{\max}$	n:100, m:20
Rinciog et al., 2020	FFc	$dp_{ji}^s, dbatch, vnops$	$f(W, T_j)$	n:10, m:15
Moser et al., 2020	Jm	$prec, recrc, s_{jk}$	$C_{\max}$	n:12, m:3
Chen et al., 2020	FJc	M_i^o	$C_{\max}$	n:20, m:15, m_{ave}^{alt}:2
Yu et al., 2020	POm	M_i^o	$C_{\max}$	n:1, m:2, o_j:100
Lang et al., 2020	$FJc + Rm$	None	$C_{\max}\| \sum T_j$	n:20, m:5, o_j:4, c:1
Martinez Jimenez et al., 2020	Jm	None	$C_{\max}$	n:5, m:5
Ren et al., 2020	Jm	None	$C_{\max}$	n:30, m:20
Li et al., 2020	Om	$vnops$	$C_{\max}$	n:10, m:10
Martins et al., 2020	$FPOc$	$fpops$	$C_{\max}$	n:10, m:7, m_{ave}^{alt}:3
Zhu et al., 2020	FFc	None	$C_{\max}$	n:20, m:50, m_{ave}^{alt}:5

Continued ...

TABLE A.1: SCHEDULING SETUPS IN LITERATURE. (Continued)

Source	Machine Setups (α)	Additional Constraints (β)	Optimization Goal (γ)	Instance Size Considered
Seito et al., 2020	Jm	None	$C_{\max}$	n:50, m:10
Moon et al., 2021	Jm	None	$C_{\max}$	n:20, m:15
Park et al., 2021	Jm	None	$C_{\max}$	n:100, m:20
Cao et al., 2021	$FPOc + Rm$	s_{jk}, M_i^o	$C_{\max}$	m:10, n:12, pcbf:3
Zhao et al., 2021a	$FPOc$	?	$C_{\max}$	n:50, m:9, o_j:20 (?)
Zhao et al., 2021c	Jm	None	$T_{\min}$	n:20, m:10
Wang et al., 2021b	Jm	None	$C_{\max}$	n:10, m:10
Pan et al., 2021	Fm	$perm$	$C_{\max}$	n:200, m:20
Han et al., 2021	FJc	M_i^o	$C_{\max}$	n:20, m:15, m_{ave}^{alt}
Cunha et al., 2021	Jm	None	$C_{\max}$	n:50,m:20
Ni et al., 2021	FFc	None	$C_{\max}$	n:240,m:5·?
Ren et al., 2021b	Fm	None	$C_{\max}$	n:80, m:5
Ren et al., 2021a	FFc	$tr(\infty)$	$C_{\max}$	?
Pol et al., 2021	$FJc + Rm$	$tr(\infty)$, M_i^o	$C_{\max}$(?)	n:600, m:6
Samsonov et al., 2021	Jm	None	$C_{\max}$	n:15, m:15
Du et al., 2022	$FJc + Rm$	M_i^o, s_{jk}, $tr(\infty)$	$f(C_{\max}, \sum EC_i)$	n:200, m:10 (?)
Long et al., 2022	FJc	M_i^o	$C_{\max}$	n:20, m:15, $\sum o_j$:232
Wang et al., 2022	FFc	$tr(\infty)$	$C_{\max}$, $\sum EC_i$	n:100, o_j:8, m_{ave}^{alt}:5
Brammer et al., 2022	POm	$perm, pops$	$C_{\max}$	n:500, m:60
Cai et al., 2022	FFc	$tr(\infty)$, $vnops$	$C_{\max}$	n:140, m:200, o_j:5, m_{ave}^{alt}:5
Kim et al., 2022	Jm	$tr(k,n)$, s_{jk}^{tr}	$C_{\max}$	n:75, m:6, r:1
		Stochastic Setups		
Mahadevan et al., 1998	Fm	$block_{out}$, dmd_j^s, $brkdwn^s$	$\sum Stk_j$| $\sum MF_i$	n:?(?), m:3
Aydin et al., 2000	Jm	r_j^s	T_{ave}	n:5(?), m:9
Wei et al., 2004	Jm	r_j	T_{ave}	n:?(15), m:9
Wang et al., 2005	1	r_j^s	$T_{\max}$| T_{ave}| $\sum U_j$	m:?(?), m:1
Paternina-Arboleda et al., 2005	1	s_{jk}, dmd_j^s	$f(Stk_j, Bf_i, \sum s_{ik}^t)$	n:?(?), m:1
Wang et al., 2007	FJc	r_j^s, p_{ji}^s, M_i^o	T_{ave}	m:5, n:?(?), o_j:3, m_{ave}^{alt}:2

Continued ...

TABLE A.1: SCHEDULING SETUPS IN LITERATURE. (Continued)

Source	Machine Setups (α)	Additional Constraints (β)	Optimization Goal (γ)	Instance Size Considered
Yingzi et al., 2009	Jm	r_j^s	$C_{\max}$	n:(?)18, m:9
Chen et al., 2010	Jm	r_j^s, $vnops$	$F_{\text{ave}}\|T_{\text{ave}}$	n:5($5e^5$), m:5
Gabel et al., 2012	Jm	p_{ji}^s	$C_{\max}$	n:30, m:15
Jiménez, 2012	FJc	p_{ji}^s, M_i^o	$C_{\max}\|T_{\text{ave}}$	n:20, o_j:15, m_{ave}^{alt}:3
Palombarini et al., 2012	1	r_j^s, $brkdwn^s$	$\sum T_j$	n:20(21), m:1
Qu et al., 2015	FFc	s_{jk}, $batch(b)$, $fmls$, dmd_j^s, $brkdwn^s$	$f(Bf_i, s_{jk}, L_j)$	n:2(?), c:3, m:6
Qu et al., 2016	FJc(?)	r_j^s, $brkdwn$, $block$, s_{jki}, j_{batch}, $vnops$, M_i^o(?)	?	n:?(?), m:6
Bouazza et al., 2017	FFc	r_j^s, s_{jk}	$\frac{1}{m}\sum w_i I_i^m$	n:9(500), m:6
Shahrabi et al., 2017	Jm	r_j^s, $brkdwn^s$	F_{ave}	n:10(?), m:10
Thomas et al., 2018	FJc	r_j^s, $tr(\infty)$, $fres$	Tpt_{ave}	n:6(?), m:6
Waschneck et al., 2018	FJc	r_j^s, M_i^o	Utl_{ave}	?
Stricker et al., 2018	FJc(?)	r_j^s, $brkdwn^s$, p_{ji}^s, $block_{in}$(?)	Utl_{ave}	n:?(?), m:3, m_{ave}^{alt}:2, 6
Shiue et al., 2018	FJc	$tr(r)$, $block_{in}$(?)	$F_{\text{ave}}\|$ $Tpt_{\text{ave}}\|$ $\sum U_j$	n:?(?), m:5, m_{ave}^{alt}:?
Kuhnle et al., 2019	FJc	r_j^s, $brkdwn^s$, p_{ji}^s, $block_{in}$(?)	$Utl_{\text{ave}}\|$ F_{ave}	n:?($450e^3$), m:3
Park et al., 2019	FJc(?)	p_{ji}^s, $recrc$, s_{jki}, $fmls$(?)	$C_{\max}$	n:12, m:?
Zhao et al., 2019	FJc	$brkdwn^s$, $recrc$, M_i^o	$C_{\max}$	n:15, m:10
Palombarini et al., 2019	Rm	r_j^s, $brkdwn^s$, j_{batch}, s_{jk}	$\sum T_j$	n:?(?), m:3, m_{ave}^{alt}:2
Han et al., 2019	$FFc + Rm$	$tr(\infty)^s$, r_j^s(?)	$C_{\max}$	n:12(100?), m:9, o_{ave}:3?
Hofmann et al., 2020	$FPOc$	r_j^s, $brkdwn^s$, M_i^o, s_{jki}, $fmls$	F_{ave}	n:10(?), c:5, m:10
Liu et al., 2020	Jm	p_{ji}^s	$C_{\max}$	n:10, m:10
Zhou et al., 2020	Rm	r_j^s, M_i^o	$C_{\max}$	n:3(?), m:5

Continued ...

TABLE A.1: SCHEDULING SETUPS IN LITERATURE. (Continued)

Source	Machine Setups (α)	Additional Constraints (β)	Optimization Goal (γ)	Instance Size Considered
Luo, 2020	FJc	r_j^s	$\sum T_j$	n:20(200), m:20
Wang, 2020	Jm	r_j^s	$\sum w_{1j}T_j + w_{2j}E_j$	n:0($3e^3$), m:8
Kuhnle et al., 2020	FJc	r_j^s, $brkdwn^s$, $recrc$, $tr(r)$, $block_{in}$	$\sum Utl_i \vert \sum I_r^{tr} \vert \sum Stk_j \vert$ $\sum Utl_r \vert Tpt_{\text{ave}}$	n:?(?), c:3, m:8?
Hu et al., 2020a	FJc(?)	$tr(r)$, r_j^s	$C_{\max}$	n:20(800), m:1
Park et al., 2020b	FJc	p_{ji}^s, $recrc$, $fmls$, M_i^o	$C_{\max}$	m:175, n:12
Wu et al., 2020	Fm	r_j^s, $prmu$, j_{batch}	$\sum w_j T_{j_{batch}}$	m:5, n:200(?)
Han et al., 2020	Jm	p_{ji}^s	$C_{\max}$	n:50, m:20
Wang et al., 2021a	$FPOc$	r_j^s, M_i^o(?)	$\sum E_j$	n:15, o_j:24, c:8, m:13
Yang et al., 2021b	Fm	r_j^s, $prmu$	$\sum T_j$	n:3(200), m:20
Yang et al., 2021a	Fm	r_j^s, $prmu$, $fmls$	$\sum w_j T_j$	n:15(150), m:8
Luo et al., 2021a	$FJc + Rm$	M_i^o, r_j^s	$f(\sum w_j T_j, Utl_{\text{ave}})$	n:20(200), m:50, m_{ave}^{alt}:25, o_j:20
Zhao et al., 2021b	Jm	r_j^s(?)	$f(C_{\max}, \sum T_j)$	?
Luo et al., 2021b	FJc	r_j^s, $brkdwn^s$, $vnops$, $pnwt$, M_i^o	$\sum_j w_j T_j$, Utl_{ave}, Bft_{std}	?
Heger et al., 2021	$Fm + Rm$	r_j^s, $fmls$, $recrc$, $tr(3)$	T_{ave}	n:(?)$1e4$, m:10
Lee et al., 2022	FFc	p_{ji}^s, M_i^o, $prmu$, $tr(r)$	$C_{\max}$	n:120,m:6, o_j:3, m_{ave}^{alt}:1
Lin et al., 2022	FJc	p_{ji}^s, s_{jki}, $recrc$, M_i^o	$C_{\max}$	n:200, m:100, o_j:3
Min et al., 2022	1	r_j^s	$\sum w_j T_j$	n:$1e3$, m:1

TABLE A.2: RL DESIGNS IN LITERATURE.

Source	**MDP Characteristics**				**Agent Characteristics**		
	MDP Breakdown	Multi Agents	State Modeling	Action Modeling	RL Class	RL Algorithm	Function Representation
	Designs for Deterministic Setups						
Zhang et al., 1995	IGI	No	F	GO (?)	v	TD(λ)	TDNN
Zhang et al., 1996	IGI	No	F	GO (?)	v	TD(λ)	TDNN
Riedmiller et al., 1999	Seq	m_p	F	PR	v	QL	FCNN
Martinez, 1999	IGI	No	R	GO	?	?	?
Csaji et al., 2003	SeR	j (?)	R	P	v	TD(λ)	Tab.
Hong et al., 2004	Seq	?	R	D	v	QL	Tab.
Gabel et al., 2007a	Seq	m_p	F (?)	D (?)	v	QL	FCNN
Gabel et al., 2007b	Seq	m_p	F (?)	D (?)	v	QL	FCNN
Gabel, 2009	Seq	m_p	F	D	v	QL	FCNN
Martínez et al., 2011	RBS	o & m_p	?	?	v	QL	Tab.
Zeng et al., 2011	Seq	m_p	F	PR	v	QL	Tab.
Petrova et al., 2013	SeR	No	?	P	?	?	?
Zhang et al., 2013	Seq	No	F	PR	v	TD(λ)	θ_{ji}
Reyna et al., 2015	Seq	m_p	F (?)	D	v	QL	Tab.
Arviv et al., 2016	Rou	m_{tr}	R	D	v	QL	Tab.
Fonseca-Reyna et al., 2018	Seq	No (?)	F (?)	D	v	QL	Tab.
Xue et al., 2018	Rou	No (?)	R	D	v	QL	Tab. (?)
Mendez-Hernandez et al., 2019	Seq	m_p	F (?)	D	v	QL	FCNN
Lin et al., 2019	Seq	No	F	PR	v	DQN	FCNN
Tan et al., 2019	Seq	m_p (?)	R	D	v	QL	Tab.
Marandi et al., 2019	SeR	No	R	P	v	QL	Tab.
Baer et al., 2019	Rou (?)	j	R	D (?)	?	?	?
Zhang et al., 2020	EdD	No	R	D	π, v	PPO	GCN
Rinciog et al., 2020	Seq	No	R	D	π, v	AZ	FCNN
Moser et al., 2020	Seq (?)	No	R (?)	D (?)	v	QL	Tab.
Chen et al., 2020	SeR	No	F	P	v	TD(λ)	Tab.
Yu et al., 2020	Seq	No	R	D	π, v	AZ	CNN
Lang et al., 2020	RBS	Subp.	R & F	D	v	DQN	FCNN & RNN
Martinez Jimenez et al., 2020	Seq	m_p	?	?	?	?	?
Ren et al., 2020	?	?	?	?	π, v	A3C	RNN
Li et al., 2020	EdD	No	R	D	π, v	VAC	GCN
Martins et al., 2020	IGI	No	F	GO	v	TD(λ)	FCNN

Continued ...

TABLE A.2: RL DESIGNS IN LITERATURE. (Continued)

Source	MDP Characteristics				Agent Characteristics		
	MDP Breakdown	Multi Agents	State Modeling	Action Modeling	RL Class	RL Algorithm	Function Representation
Zhu et al., 2020	IGI (?)	No	R	D	π, v	PPO	FCNN
Seito et al., 2020	EdD	No	R	D	?	?	GCN
Moon et al., 2021	Seq	No	F	PR	v	DQN	FCNN
Park et al., 2021	Seq	No	R	D	π, v	PPO	GCN
Cao et al., 2021	SeR	No	F	P	v	SARSA	Tab.
Zhao et al., 2021a	SeR	?	F (?)	P	v	QL (?)	Tab. (?)
Zhao et al., 2021c	Seq	No	F	PR	v	DQN	FCNN
Wang et al., 2021b	Seq	No	R	D	π, v	PPO	?
Pan et al., 2021	DiP	No	R	D	π, v	VAC	RNN
Han et al., 2021	RBS (?)	No	R (?)	D	π, v	VAC	RNN
Cunha et al., 2021	Seq (?)	No	R (?)	D (?)	?	?	?
Ni et al., 2021	SeR (?)	No	R	D	π, v	PPO	GCN
Ren et al., 2021b	Seq (?)	No	F	PR	v	SARSA	FCNN
Ren et al., 2021a	Seq	m_p (?)	R (?)	D (?)	v	QL	Tab.
Pol et al., 2021	TCS	No	R (?)	D (?)	v	DQN	FCNN
Samsonov et al., 2021	Seq	No	F	P	v	DQN	FCNN (?)
Du et al., 2022	IGI	No	F	GO	v	DQN	FCNN
Long et al., 2022	SeR	No	F	P	v	QL	Tab.
Wang et al., 2022	IGI	No	F	GO	π	REINF.	FCNN + Att.
Brammer et al., 2022	Seq	No	F	D	π, v	PPO	?
Cai et al., 2022	SeR	No	F	P	v	QL	Tab.
Kim et al., 2022	TCS	No	R (?)	D	v	QL	FCNN (?)
	Designs for Stochastic Setups						
Mahadevan et al., 1998	Seq	No	?	?	v	?	?
Aydin et al., 2000	Seq	No	F	PR	v	QL	FCNN
Wei et al., 2004	Seq (?)	No	F	PR	v	QL	Tab.
Wang et al., 2005	Seq	No	F	PR	v	QL	Tab.
Paternina-Arboleda et al., 2005	Seq	j	F	D	v	QL	FCNN
Wang et al., 2007	Rou	No (?)	F	D	v	QL	Tab.
Yingzi et al., 2009	Seq	No	F	PR	v	QL	Tab.
Chen et al., 2010	Seq	No	F	PR	v	QL	Tab.
Gabel et al., 2012	Seq	m_p	R	D	π	PG	FCNN
Jiménez, 2012	IRS	m_p	F (?)	D	v	QL	FCNN

Continued ...

TABLE A.2: RL DESIGNS IN LITERATURE. (Continued)

Source	**MDP Characteristics**				**Agent Characteristics**		
	MDP Breakdown	Multi Agents	State Modeling	Action Modeling	RL Class	RL Algorithm	Function Representation
Palombarini et al., 2012	IGT	No	R	GO	v	QL	Tab.
Qu et al., 2015	Seq	m_p	R	D	v	QL	Tab.
Qu et al., 2016	Seq	No	F	D (?)	v	QL	?
Bouazza et al., 2017	IRS	fml, m_p	R	PR	v	QL	Tab.
Shahrabi et al., 2017	ReS	No	F	P	v	QL	Tab.
Thomas et al., 2018	Seq	No	R & F	D	v	QL	FCNN (?)
Waschneck et al., 2018	Seq	m_p	R (?)	?	v	QL	FCNN
Stricker et al., 2018	Rou	No	R	D	v	QL	FCNN
Shiue et al., 2018	(?)	No	F	PR	v	QL	Tab. + SOM
Kuhnle et al., 2019	Rou	No	R	D	π	TRPO	FCNN
Park et al., 2019	Seq	No	R & F	D?	v	DDQN	FCNN
Zhao et al., 2019	Seq (?)	No	F (?)	PR	v	QL	Tab.
Palombarini et al., 2019	IGT	No	R	GO	v	DQN	CNN
Han et al., 2019	Rou	j	R	D	v	QL	Tab.
Hofmann et al., 2020	Rou	No	F	D	v	DQN	FCNN
Liu et al., 2020	Seq	m_p	R	PR	π, v	A2C	CNN
Zhou et al., 2020	Seq (?)	No (?)	F	D	v	DDQN	FCNN (?)
Luo, 2020	Seq	No	F	PR	v	DDQN	FCNN
Wang, 2020	Seq	m_p	F	PR	v	QL	FCNN (?)
Kuhnle et al., 2020	TCS	No	R & F	D	π	TRPO	FCNN
Hu et al., 2020a	TCS (?)	No	F	PR (?)	v	DDQN	FCNN
Park et al., 2020b	Seq	m_p	R	D	v	DDQN	FCNN
Wu et al., 2020	DiP	No	R	D	π	REINF.	RNN
Han et al., 2020	Seq	No	R	PR	v	DDDQN	FCNN
Wang et al., 2021a	IRS	Subp.	F	PR	v	QL	Tab.
Yang et al., 2021b	Seq	No	F	PR	π, v	A2C	FCNN
Yang et al., 2021a	ITS	Subp.	F	PR	π, v	A2C	FCNN
Luo et al., 2021a	IRS	No	F	PR	v	DDQN	FCNN
Zhao et al., 2021b	Seq (?)	No	R	PR	v	DDQN	CNN
Luo et al., 2021b	IRS	Subp.	F	PR	π, v	PPO	FCNN
Heger et al., 2021	Seq	No	F	P	v	?	?
Lee et al., 2022	Rou	No (?)	F	D	v	QL	Tab.
Lin et al., 2022	Seq	No	F	D	v	QL	Tab.

Continued ...

Table A.2: RL Designs in Literature. (Continued)

Source	MDP Characteristics				Agent Characteristics		
	MDP Breakdown	Multi Agents	State Modeling	Action Modeling	RL Class	RL Algorithm	Function Representation
Min et al., 2022	Seq	No	F	P	π, v	DDPG	CNN

Table A.3: Validation of RL Experiments in Literature.

Source	Reproducibility					Evaluation		
	Setup Clarity	RL Clarity	Code Availability	Input Availability	Reproducible Stochasticity	Train-Test Split	Sufficient Baselines	Cherry Picking
	Deterministic Setups							
Zhang et al., 1995	●	◐	○	○	N/A	●	●	◐
Zhang et al., 1996	●	◐	○	○	N/A	●	●	◐
Riedmiller et al., 1999	◐	●	○	○	N/A	●	◐	○
Martinez, 1999	◐	◐	○	○	N/A	○	○	●
Csaji et al., 2003	◐	◐	○	○	N/A	○	○	●
Hong et al., 2004	●	●	○	○	N/A	○	◐	●
Gabel et al., 2007a	●	◐	○	●	N/A	●	◐	○
Gabel et al., 2007b	●	◐	○	●	N/A	●	◐	○
Gabel, 2009	●	●	○	●	N/A	●	●	○
Martínez et al., 2011	●	◐	○	●	N/A	○	●	○
Zeng et al., 2011	●	●	○	○	N/A	○	●	◐
Petrova et al., 2013	●	○	○	●	N/A	○	◐	◐
Zhang et al., 2013	●	●	○	●	N/A	○	●	○
Reyna et al., 2015	●	◐	○	●	N/A	○	●	○
Arviv et al., 2016	●	●	○	○	N/A	○	●	◐
Fonseca-Reyna et al., 2018	●	◐	○	●	N/A	○	●	○
Xue et al., 2018	●	◐	○	○	N/A	○	○	◐
Mendez-Hernandez et al., 2019	●	◐	○	●	N/A	○	●	○
Lin et al., 2019	●	◐	○	●	N/A	○	◐	◐
Tan et al., 2019	◐	◐	○	○	N/A	○	○	●
Marandi et al., 2019	●	◐	○	○	N/A	○	●	○
Baer et al., 2019	○	◐	○	○	N/A	○	○	●
Zhang et al., 2020	●	●	○	◐	N/A	●	●	○
Rinciog et al., 2020	●	●	○	○	N/A	●	◐	◐

Continued ...

TABLE A.3: VALIDATION OF RL EXPERIMENTS IN LITERATURE. (Continued)

Source	**Reproducibility**					**Evaluation**		
	Setup Clarity	RL Clarity	Code Availability	Input Availability	Reproducible Stochasticity	Train-Test Split	Sufficient Baselines	Cherry Picking
Moser et al., 2020	●	◐	○	○	N/A	○	●	○
Chen et al., 2020	●	●	○	●	N/A	○	◐	○
Yu et al., 2020	●	●	○	○	N/A	○	◐	●
Lang et al., 2020	◐	●	○	●	N/A	○	◐	◐
Martinez Jimenez et al., 2020	●	○	○	●	N/A	○	◐	●
Ren et al., 2020	●	◐	○	●	N/A	○	◐	◐
Li et al., 2020	●	●	○	○	N/A	●	●	○
Martins et al., 2020	●	●	○	○	N/A	○	●	●
Zhu et al., 2020	●	◐	○	○	N/A	○	●	◐
Seito et al., 2020	●	○	○	◐	N/A	●	○	●
Moon et al., 2021	●	◐	○	●	N/A	○	◐	●
Park et al., 2021	●	●	●	●	N/A	●	●	○
Cao et al., 2021	●	●	○	●	N/A	○	◐	○
Zhao et al., 2021a	◐	○	○	○	N/A	○	◐	◐
Zhao et al., 2021c	●	●	○	◐	N/A	○	◐	◐
Wang et al., 2021b	◐	◐	○	○	N/A	○	●	◐
Pan et al., 2021	●	●	○	●	N/A	●	◐	○
Han et al., 2021	●	◐	○	◐	N/A	●	◐	○
Cunha et al., 2021	●	◐	○	●	N/A	○	◐	○
Ni et al., 2021	●	●	○	◐	N/A	○	●	○
Ren et al., 2021b	●	◐	○	◐	N/A	●	●	◐
Ren et al., 2021a	○	○	○	○	N/A	○	◐	●
Pol et al., 2021	●	○	○	○	N/A	○	◐	●
Samsonov et al., 2021	●	●	○	●	N/A	○	●	◐
Du et al., 2022	◐	●	○	○	N/A	○	●	○
Wang et al., 2022	●	●	○	○	N/A	●	◐	◐
Long et al., 2022	●	●	○	●	N/A	○	●	○
Brammer et al., 2022	●	◐	○	◐	N/A	○	●	○
Cai et al., 2022	●	●	○	○	N/A	○	◐	○
Kim et al., 2022	●	◐	○	○	N/A	○	◐	◐
	Stochastic Setups							
Mahadevan et al., 1998	◐	◐	○	○	○	○	◐	●
Aydin et al., 2000	◐	●	○	○	○	○	◐	◐
Wei et al., 2004	◐	◐	○	○	○	○	◐	●
Wang et al., 2005	●	●	○	○	○	○	◐	◐

Continued ...

TABLE A.3: VALIDATION OF RL EXPERIMENTS IN LITERATURE. (Continued)

Source	Reproducibility					Evaluation		
	Setup Clarity	RL Clarity	Code Availability	Input Availability	Reproducible Stochasticity	Train-Test Split	Sufficient Baselines	Cherry Picking
Paternina-Arboleda et al., 2005	●	●	○	○	○	○	◐	◐
Wang et al., 2007	◐	◐	○	○	○	○	◐	◐
Yingzi et al., 2009	●	●	○	○	○	○	◐	◐
Chen et al., 2010	●	●	○	○	○	●	◐	○
Gabel et al., 2012	●	◐	○	◐	○	○	◐	○
Jiménez, 2012	◐	◐	○	◐	○	○	◐	◐
Palombarini et al., 2012	●	◐	○	○	○	○	○	◐
Qu et al., 2015	●	●	○	○	○	●	◐	◐
Qu et al., 2016	◐	◐	○	○	○	○	◐	○
Bouazza et al., 2017	◐	◐	○	○	○	○	◐	◐
Shahrabi et al., 2017	◐	●	○	○	○	○	◐	◐
Thomas et al., 2018	◐	●	○	○	○	○	◐	◐
Waschneck et al., 2018	◐	◐	○	○	○	○	◐	◐
Stricker et al., 2018	◐	●	○	○	○	○	○	◐
Shiue et al., 2018	○	○	○	○	○	○	◐	◐
Kuhnle et al., 2019	◐	●	○	○	○	○	○	○
Park et al., 2019	◐	◐	○	○	○	●	●	○
Zhao et al., 2019	●	○	○	○	○	○	◐	●
Palombarini et al., 2019	◐	●	○	○	○	○	◐	●
Han et al., 2019	◐	●	○	○	○	●	◐	●
Hofmann et al., 2020	●	●	○	○	○	○	◐	○
Liu et al., 2020	●	●	○	●	○	○	●	○
Zhou et al., 2020	◐	◐	○	○	○	○	◐	◐
Luo, 2020	●	●	○	○	○	○	◐	◐
Wang, 2020	●	◐	○	○	○	●	○	◐
Kuhnle et al., 2020	●	●	●	○	◐	●	◐	○
Hu et al., 2020a	●	●	○	○	○	○	◐	◐
Park et al., 2020b	●	●	○	○	○	●	◐	◐
Wu et al., 2020	●	●	○	○	○	●	●	○
Han et al., 2020	●	●	○	◐	○	●	●	○
Wang et al., 2021a	◐	◐	○	○	○	○	◐	◐
Yang et al., 2021b	●	●	○	○	○	●	●	○
Yang et al., 2021a	●	●	○	○	○	●	●	◐
Luo et al., 2021a	●	●	○	○	○	●	◐	○

Continued ...

TABLE A.3: VALIDATION OF RL EXPERIMENTS IN LITERATURE. (Continued)

Source	Reproducibility					Evaluation		
	Setup Clarity	RL Clarity	Code Availability	Input Availability	Reproducible Stochasticity	Train-Test Split	Sufficient Baselines	Cherry Picking
Zhao et al., 2021b	◐	◐	○	○	○	●	◐	●
Luo et al., 2021b	●	●	○	○	○	●	◐	◐
Heger et al., 2021	●	◐	○	○	○	●	◐	○
Lee et al., 2022	●	●	○	○	○	○	◐	○
Lin et al., 2022	◐	●	○	◐	○	○	●	◐
Min et al., 2022	●	●	○	○	○	●	◐	◐

Appendix B

FabricatioRL Classes

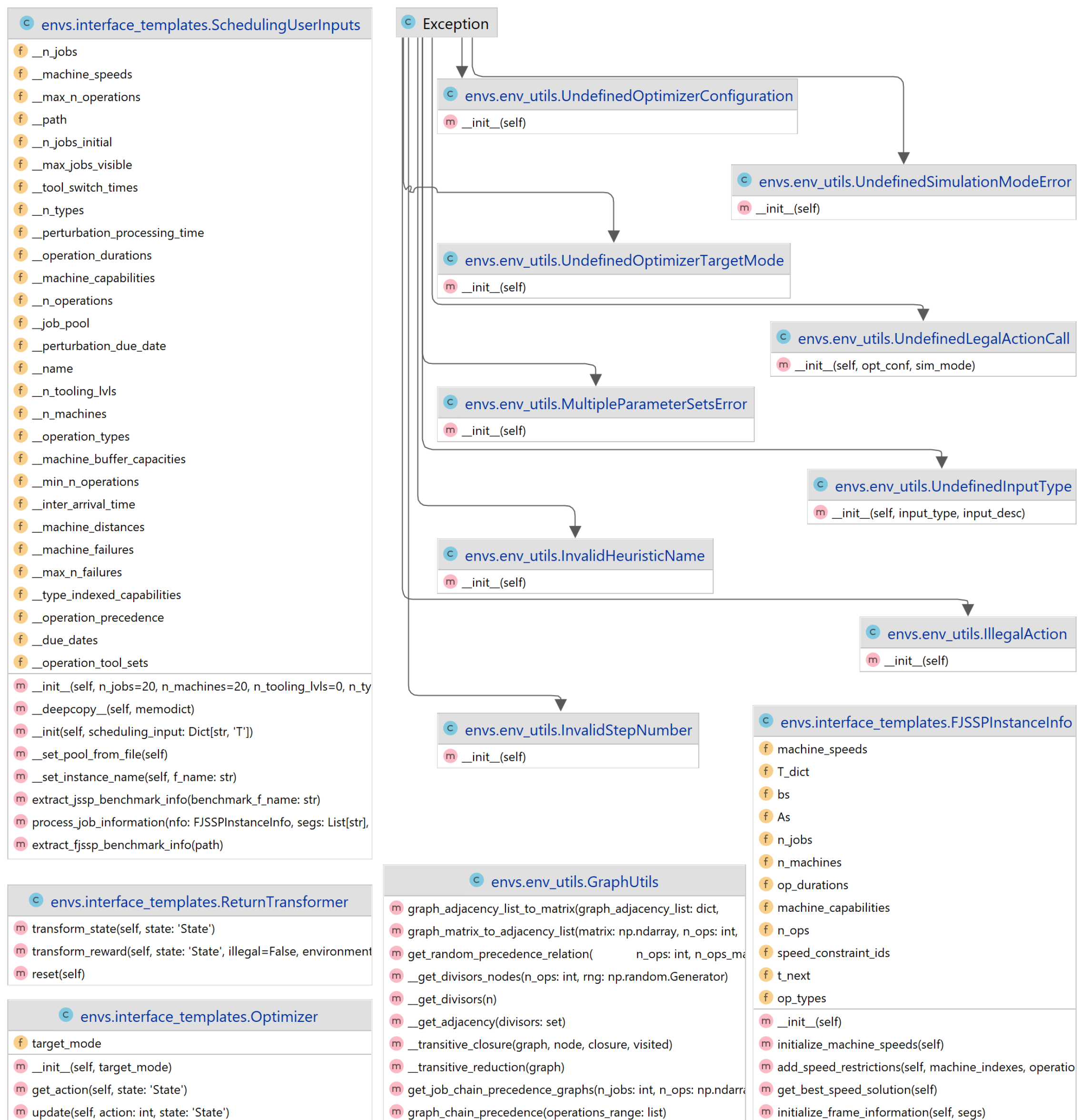


Figure B.1: interface_templates and env_utils Module Classes.

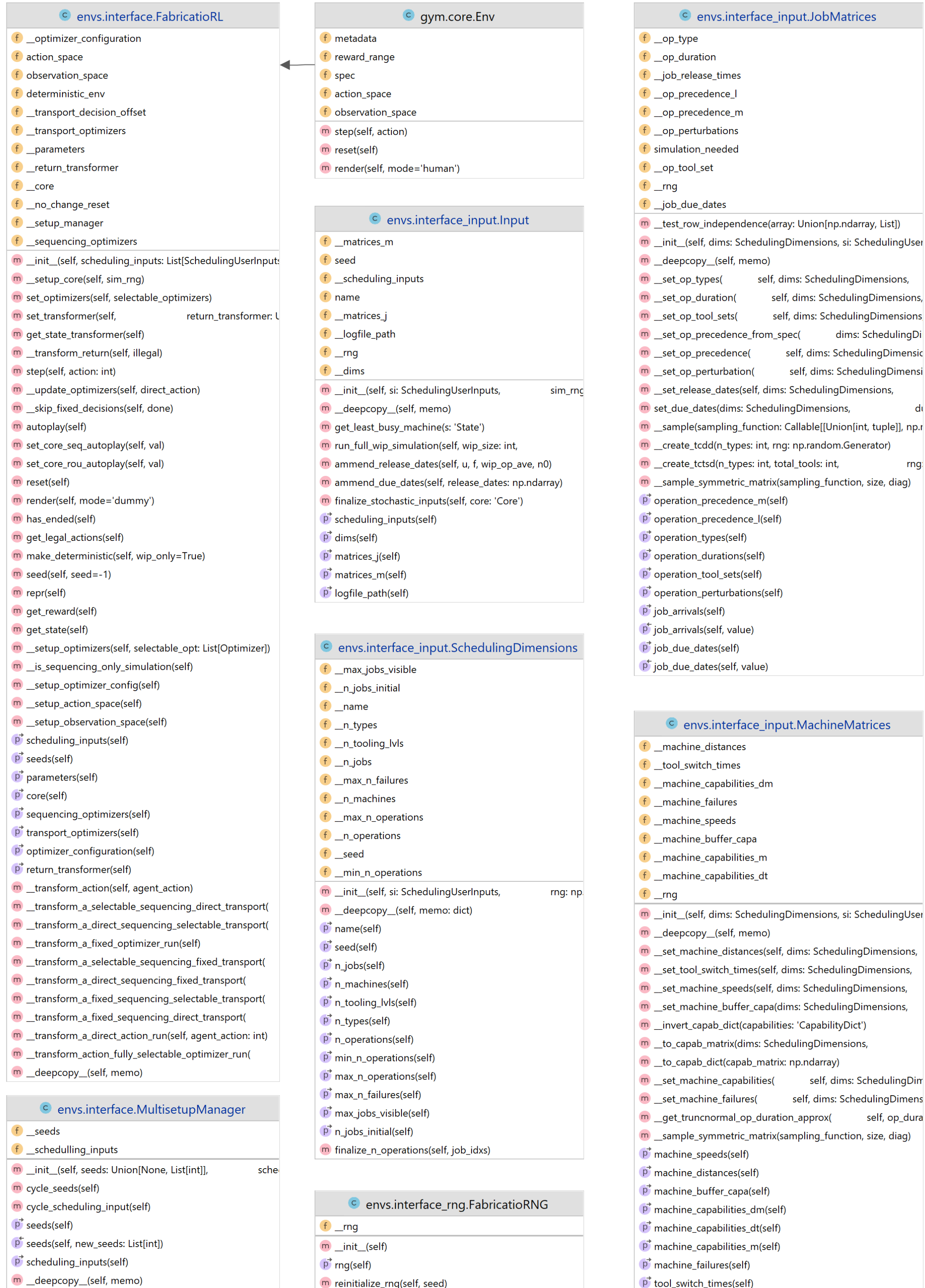


Figure B.2: interface, interface_input, and interface_RNG Module Classes.

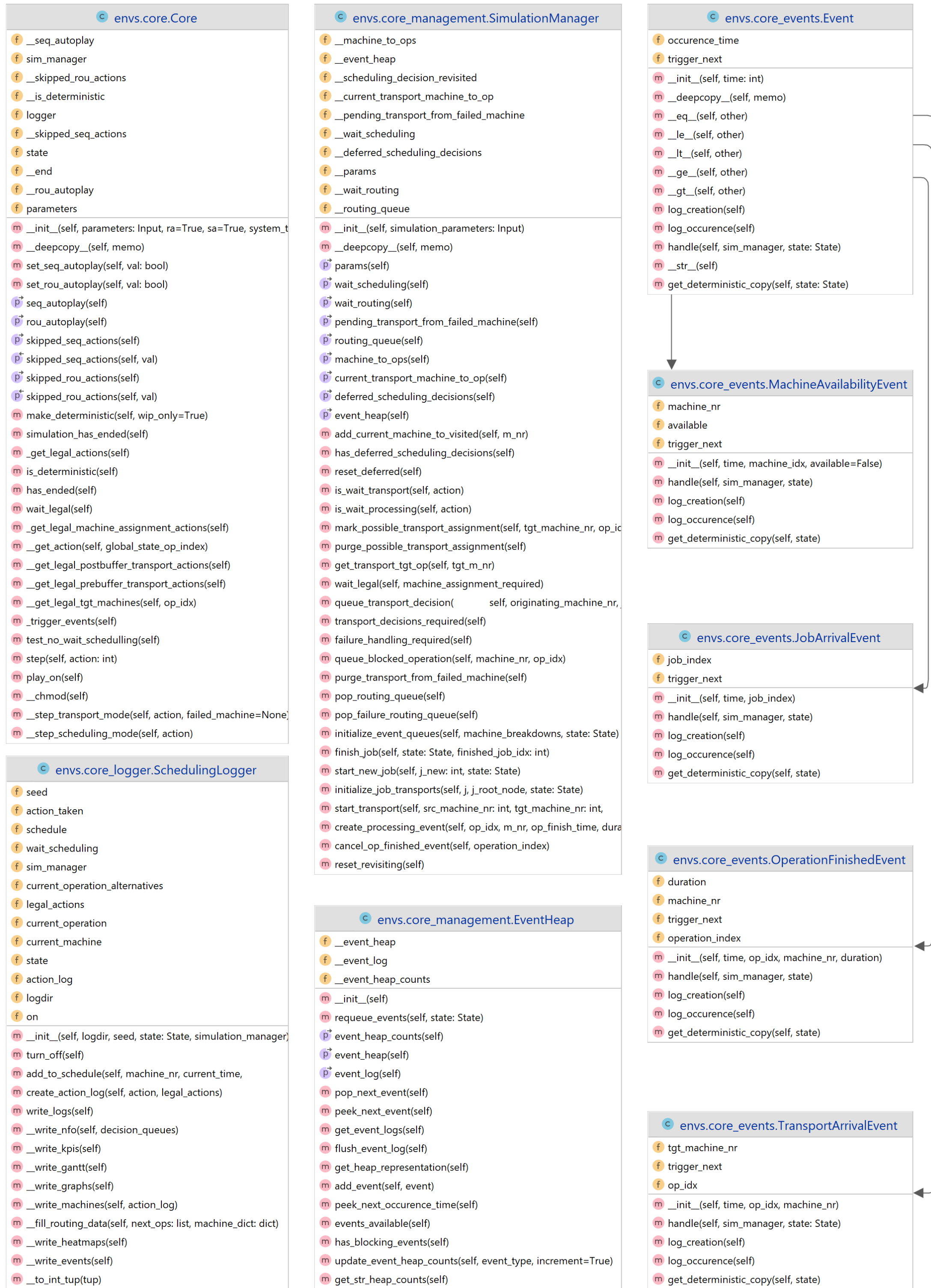


Figure B.3: core, events, core_management and logger Module Classes.

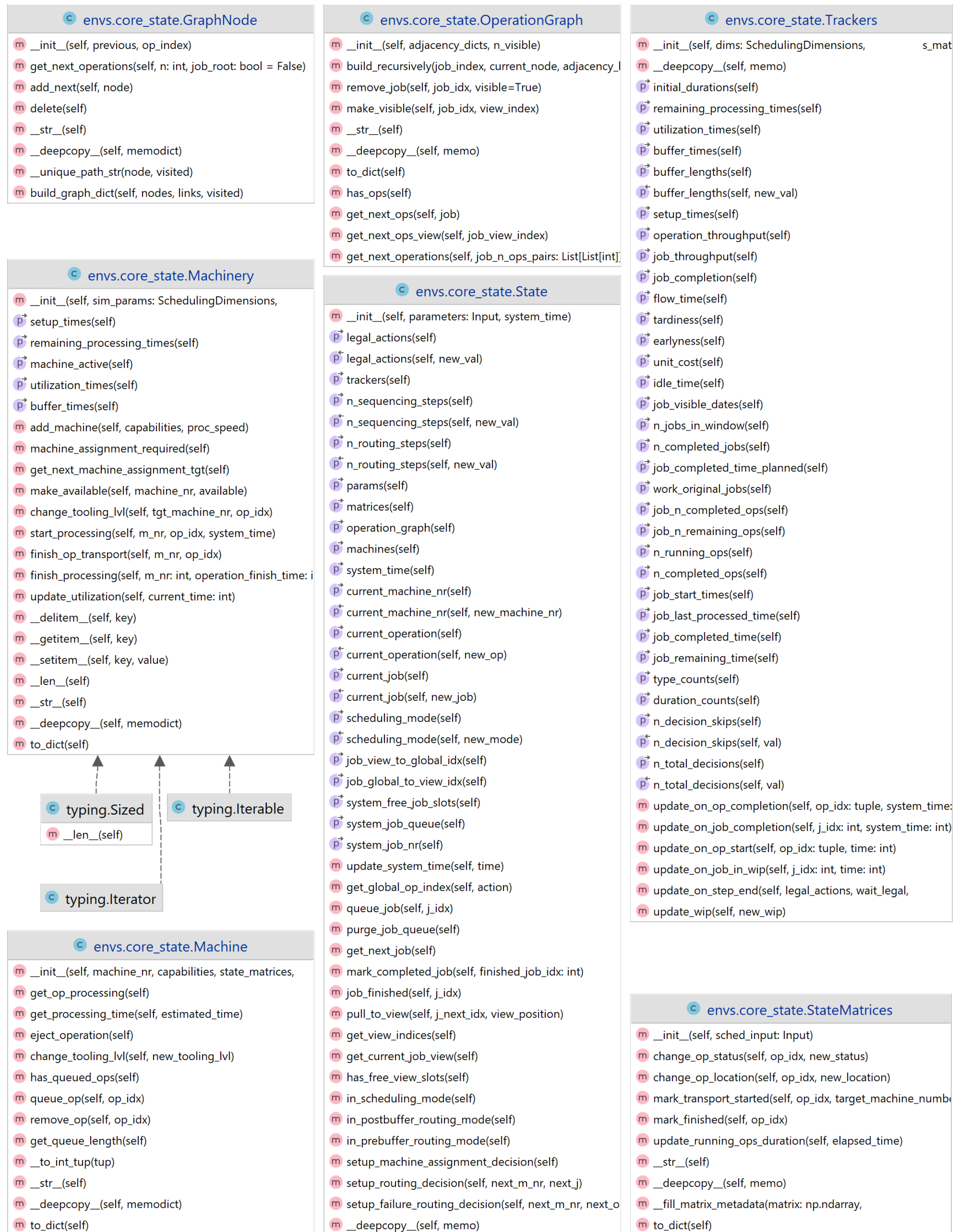
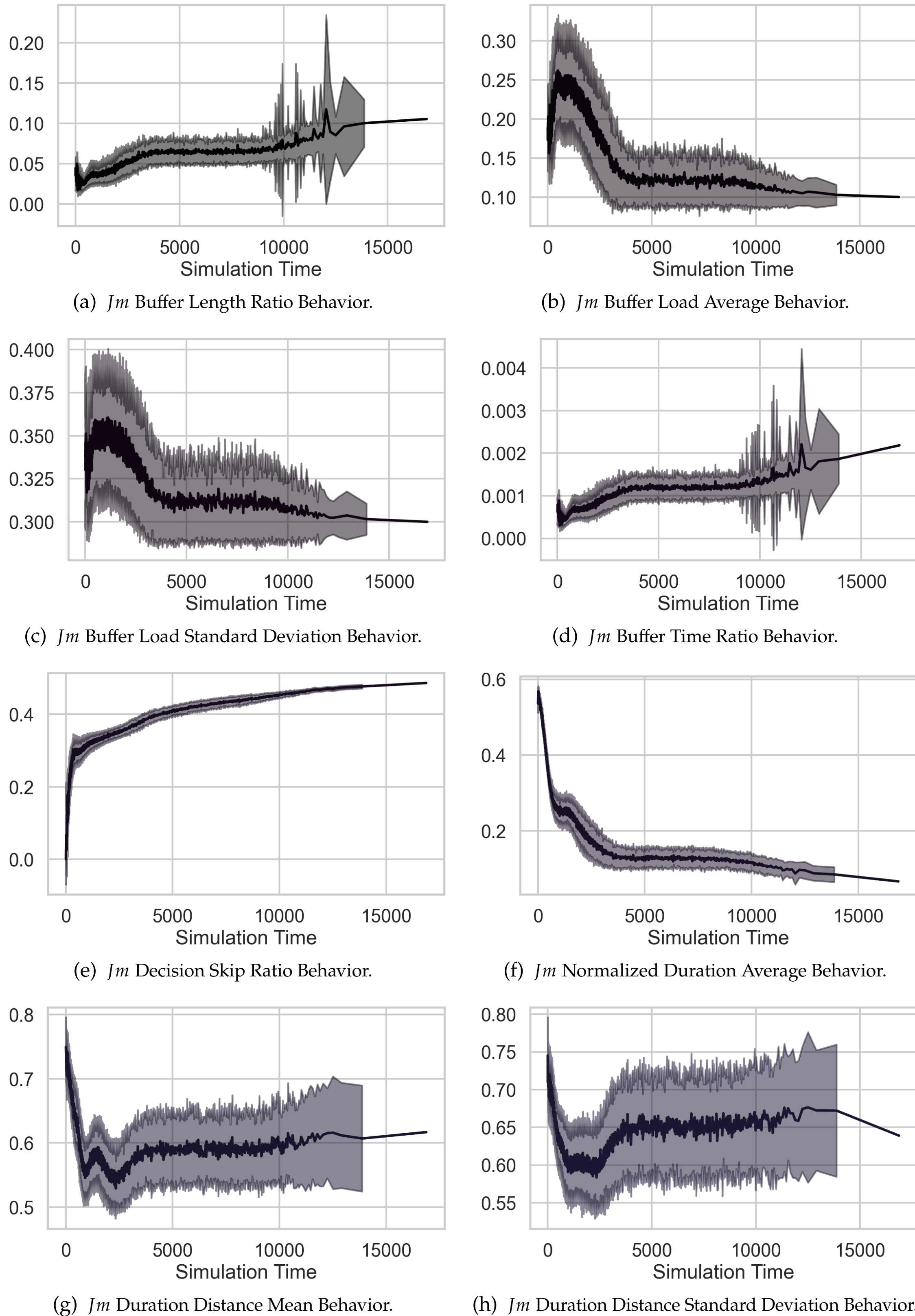


Figure B.4: state Module Classes.

Appendix C

Feature Evolution Over Time



(a) Jm Buffer Length Ratio Behavior.

(b) Jm Buffer Load Average Behavior.

(c) Jm Buffer Load Standard Deviation Behavior.

(d) Jm Buffer Time Ratio Behavior.

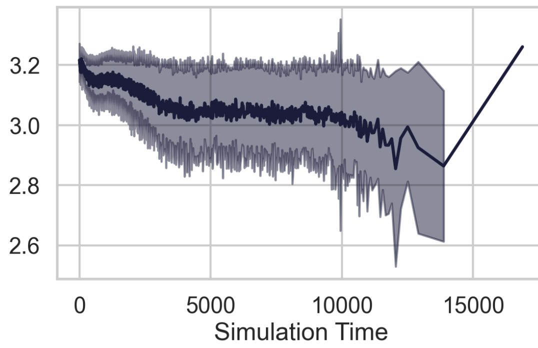
(e) Jm Decision Skip Ratio Behavior.

(f) Jm Normalized Duration Average Behavior.

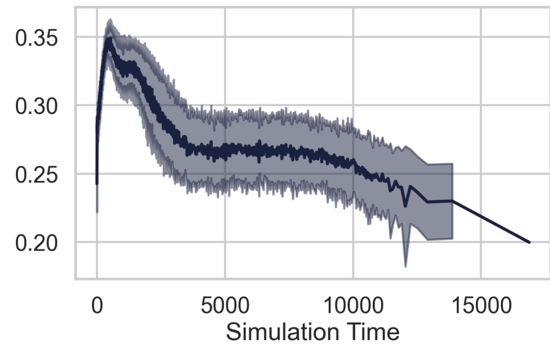
(g) Jm Duration Distance Mean Behavior.

(h) Jm Duration Distance Standard Deviation Behavior.

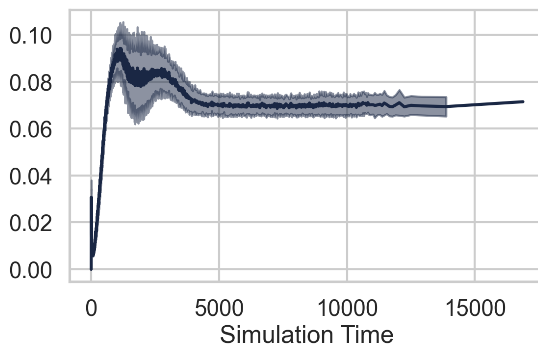
Figure C.1: Jm Feature Behavior (1-8).



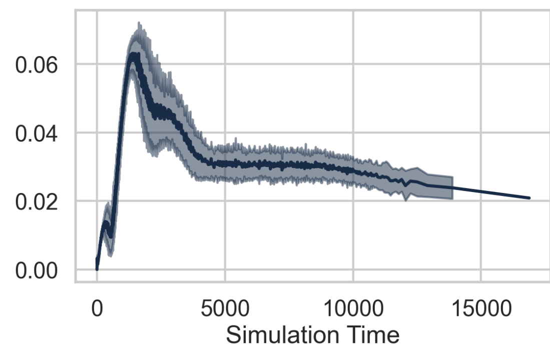
(a) Jm Duration Entropy Behavior.



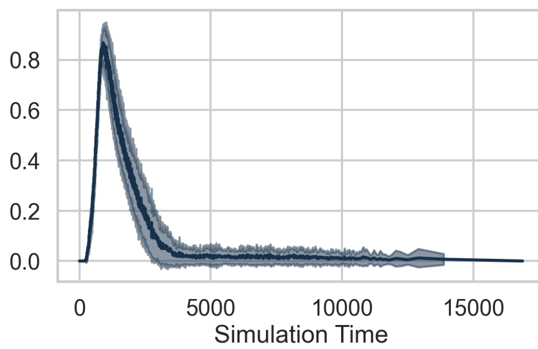
(b) Jm Normalized Duration Standard Deviation Behavior.



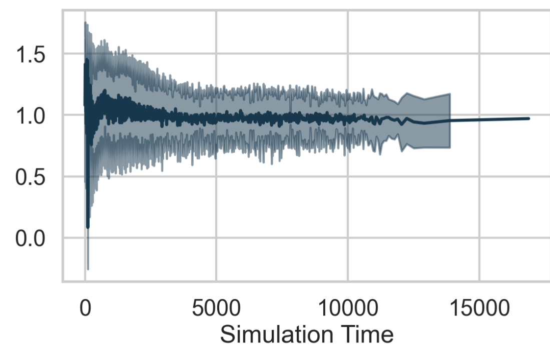
(c) Jm Estimated Flow Time Average Behavior.



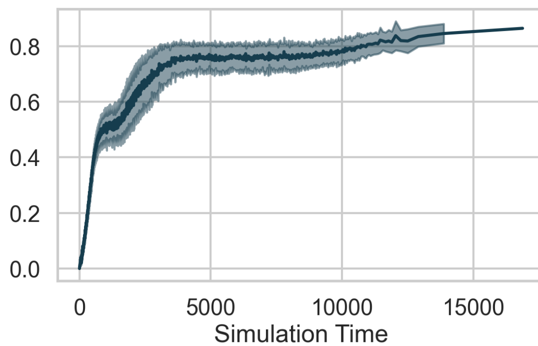
(d) Jm Estimated Flow Time Standard Deviation Behavior.



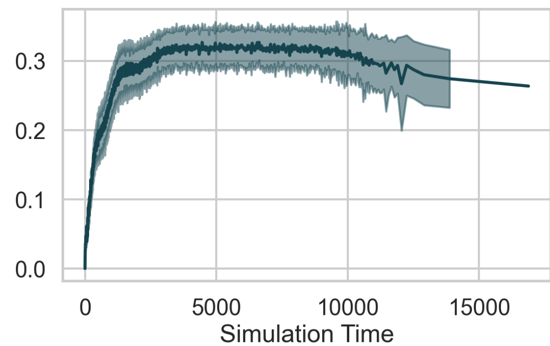
(e) Jm Estimated Tardiness Rate Behavior.



(f) Jm Heuristic Agreement Entropy Behavior.

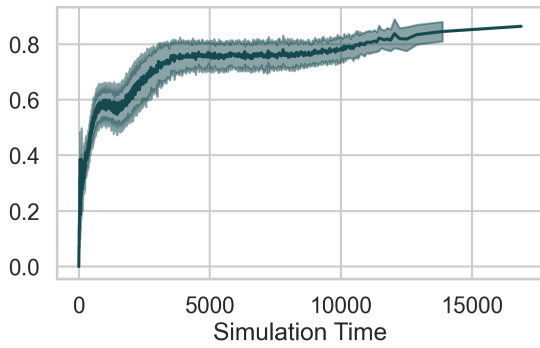


(g) Jm Job Operation Completion Rate Average Behavior.

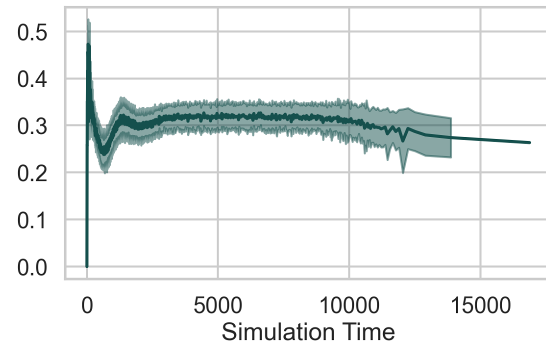


(h) Jm Job Operation Completion Rate Standard Deviation Behavior.

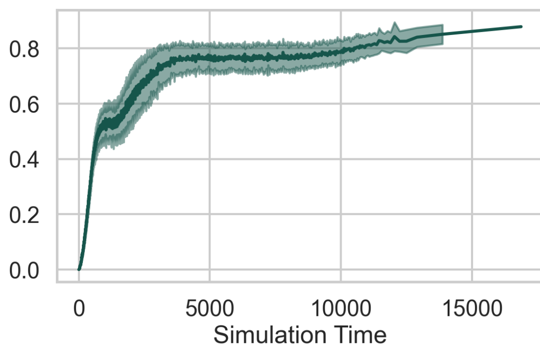
Figure C.2: Jm Feature Behavior (9-16).



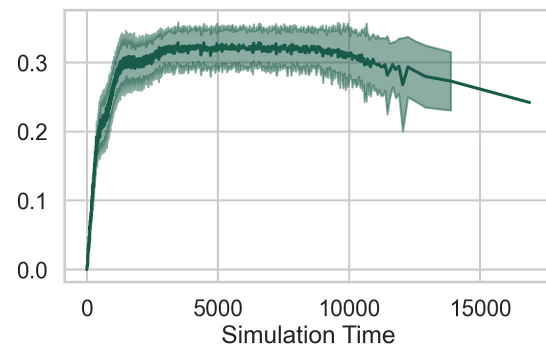
(a) *Jm* Job Operation Max Relative Completion Rate Average Behavior.



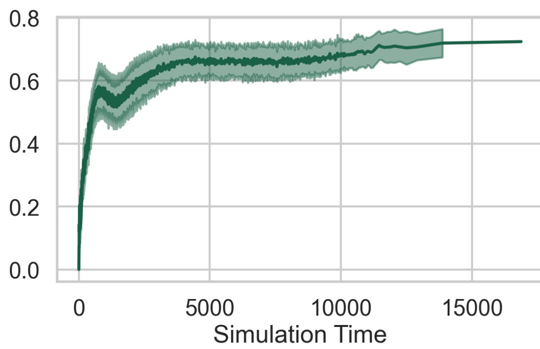
(b) *Jm* Job Operation Max Relative Completion Rate Standard Deviation Behavior.



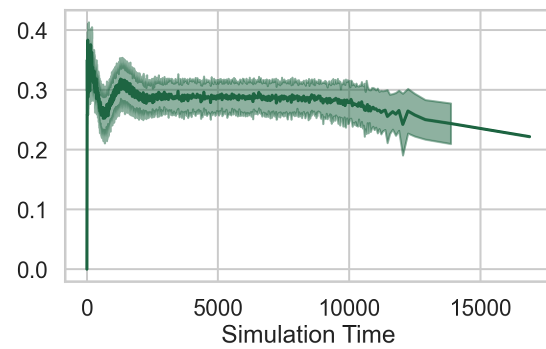
(c) *Jm* Job Work Completion Rate Average Behavior.



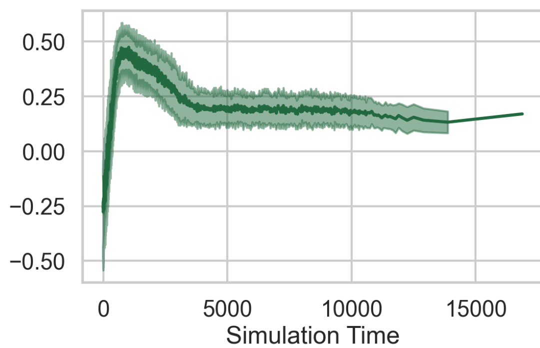
(d) *Jm* Job Work Completion Rate Standard Deviation Behavior.



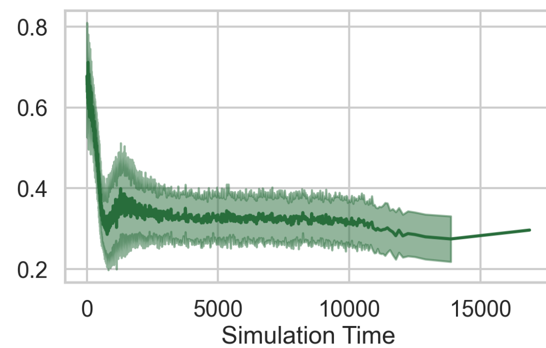
(e) *Jm* Max Relative Work Completion Rate Average Behavior.



(f) *Jm* Max Relative Work Completion Rate Standard Deviation Behavior.

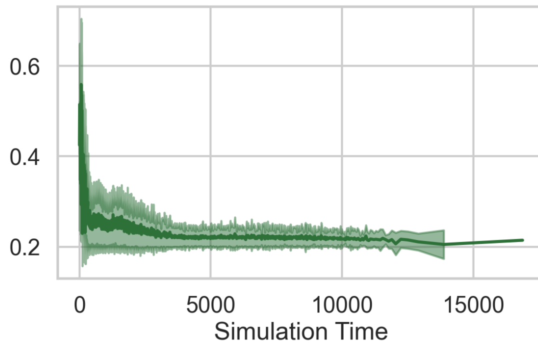


(g) *Jm* Kendall Tau Average Behavior.

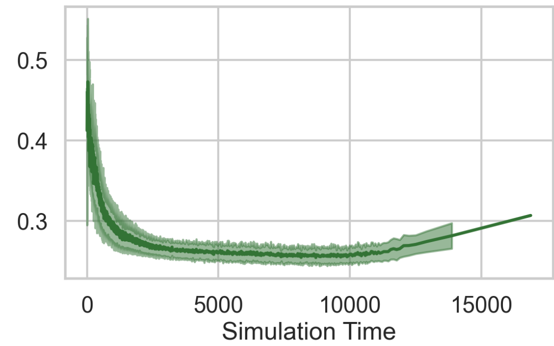


(h) *Jm* Kendall Tau Standard Deviation Behavior.

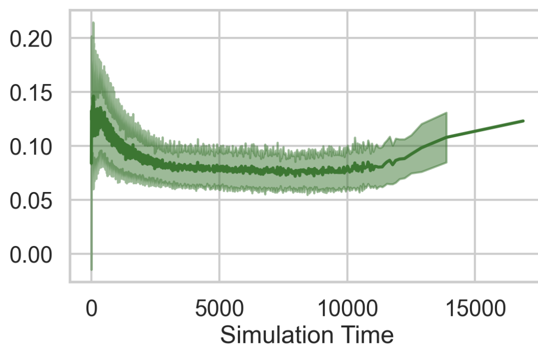
Figure C.3: *Jm* Feature Behavior (17-24).



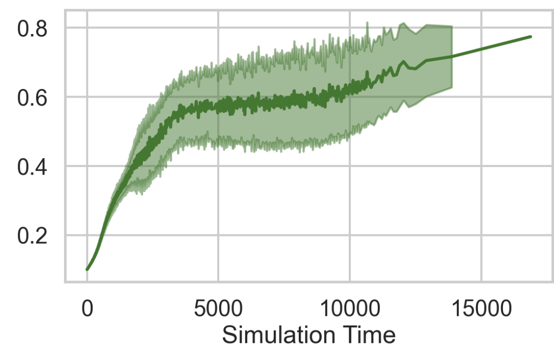
(a) Jm Legal Action to Job Ratio Behavior.



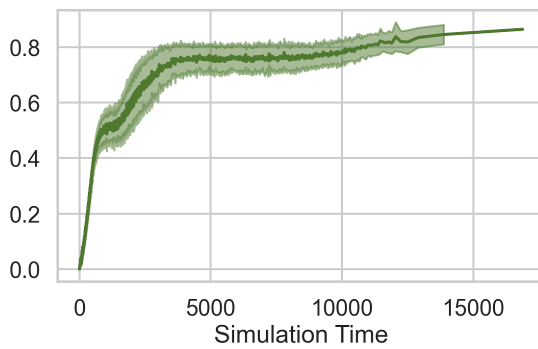
(b) Jm Legal Action Length Stream Average Behavior.



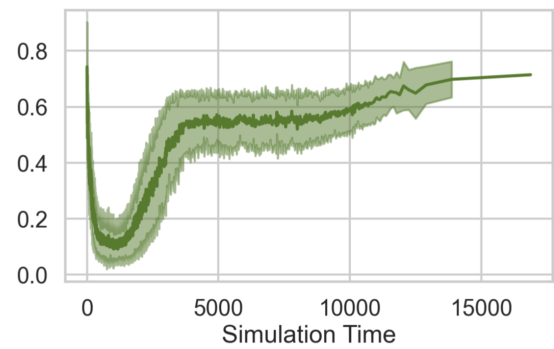
(c) Jm Legal Action Length Stream Standard Deviation Behavior.



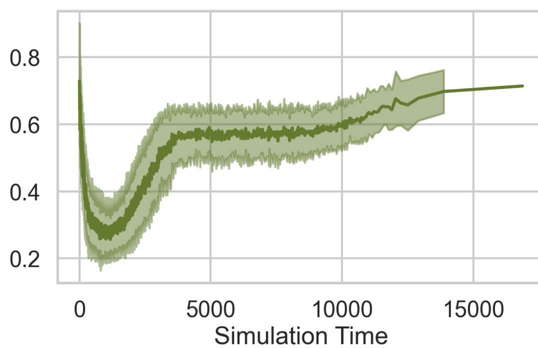
(d) Jm Makespan Lower Bound to Upper Bound Ratio Behavior.



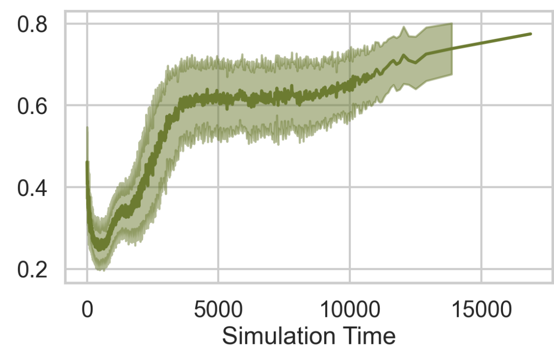
(e) Jm Operation Completion Rate Behavior.



(f) Jm Silhouette Maximum k Behavior.

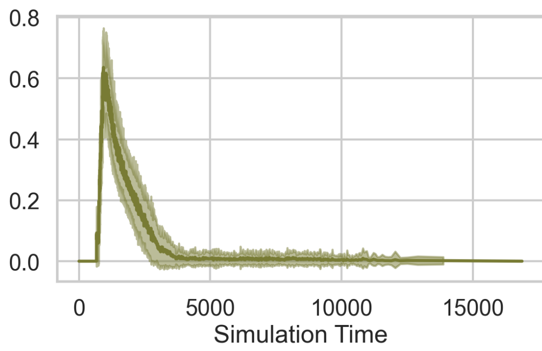


(g) Jm Silhouette Mid k Behavior.

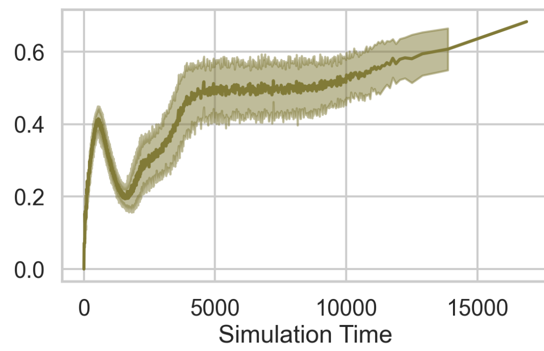


(h) Jm Silhouette Minimum k Behavior.

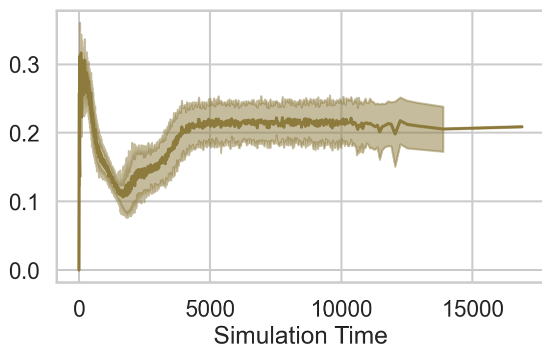
Figure C.4: Jm Feature Behavior (25-32).



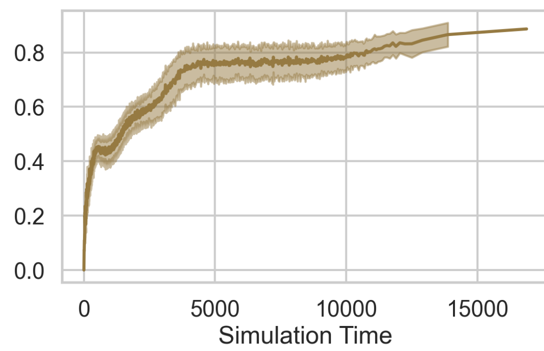
(a) Jm Estimated Tardiness Rate Behavior.



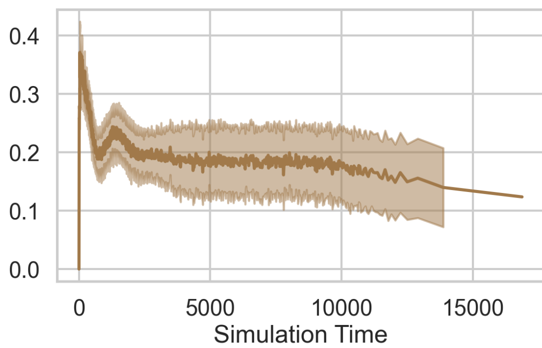
(b) Jm Absolute Job Throughput Time Average Behavior.



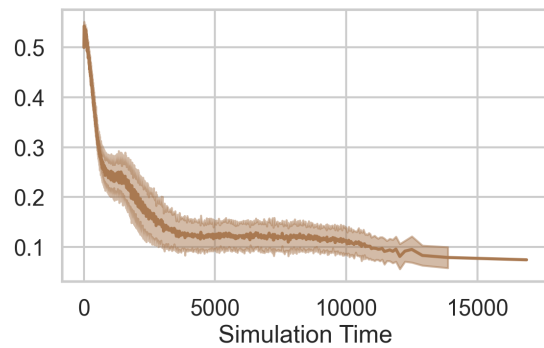
(c) Jm Absolute Job Throughput Time Standard Deviation Behavior.



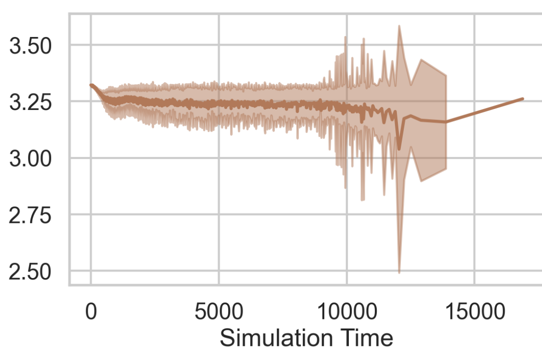
(d) Jm Relative Job Throughput Time Average Behavior.



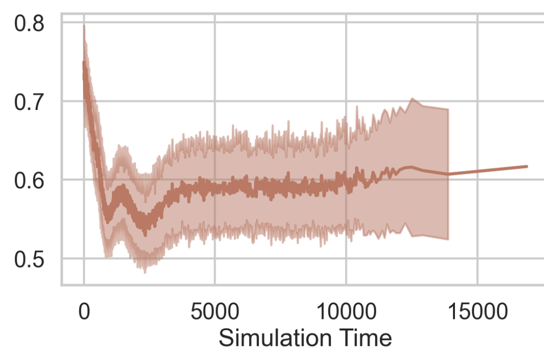
(e) Jm Relative Job Throughput Time Standard Deviation Behavior.



(f) Jm Normalized Type Average Behavior.

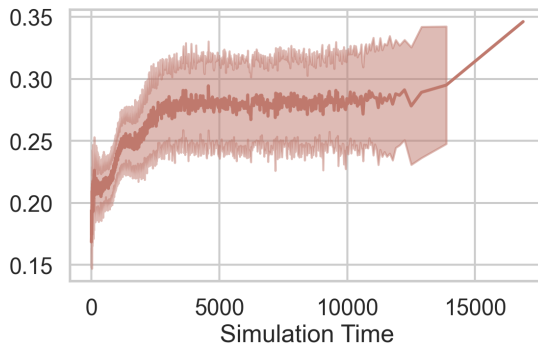


(g) Jm Type Entropy Behavior.

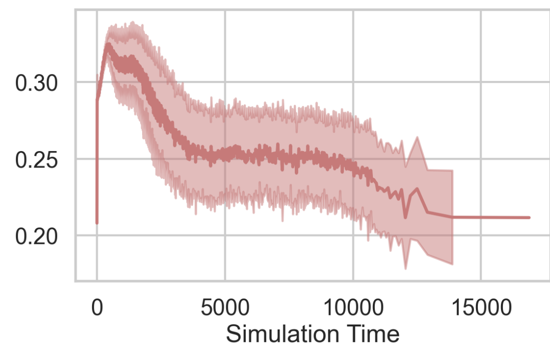


(h) Jm Type Hamming Mean Behavior.

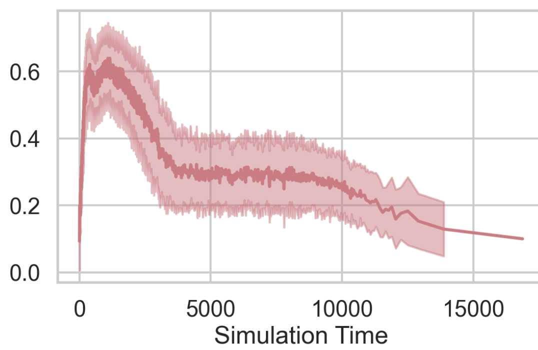
Figure C.5: Jm Feature Behavior (33-40).



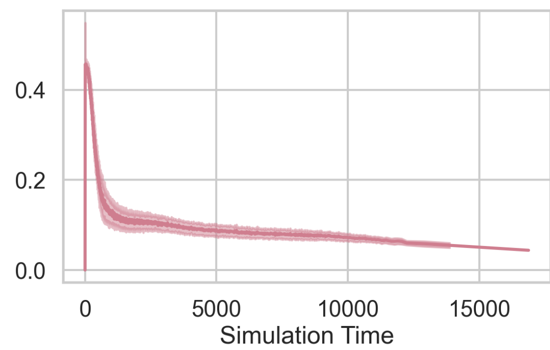
(a) Jm Type Hamming Standard Deviation Behavior.



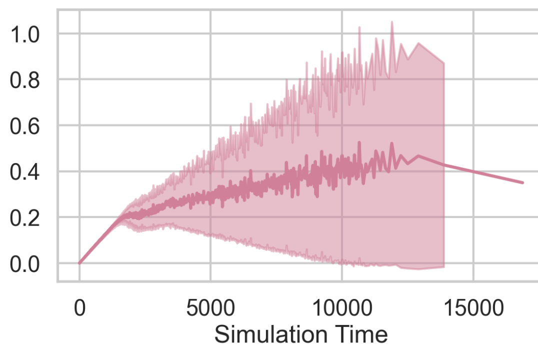
(b) Jm Normalized Type Standard Deviation Behavior.



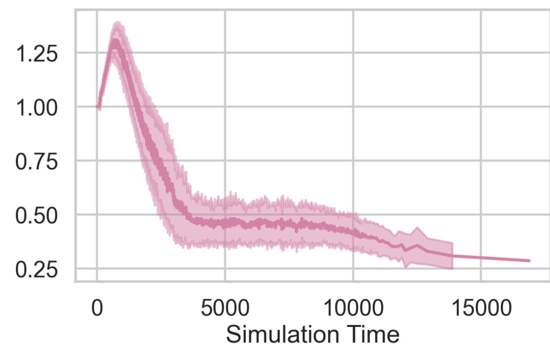
(c) Jm Current Utilization Behavior.



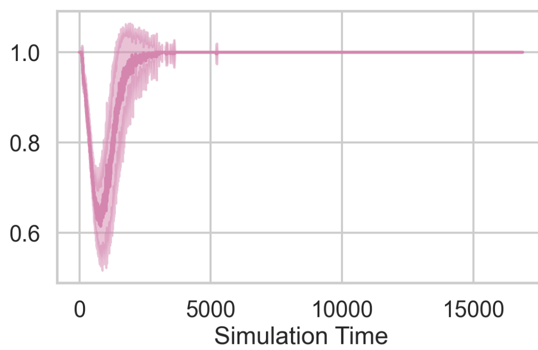
(d) Jm Utilization Standard Deviation Behavior.



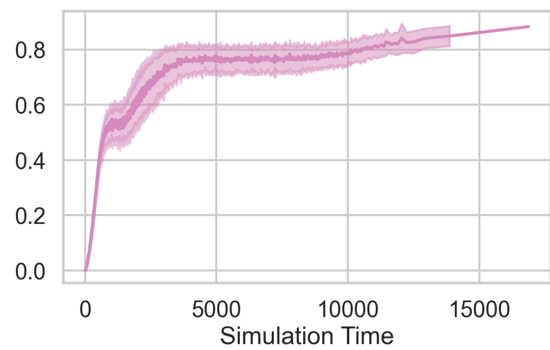
(e) Jm WIP Relative System Time Behavior.



(f) Jm WIP to Arrival Ratio Behavior.

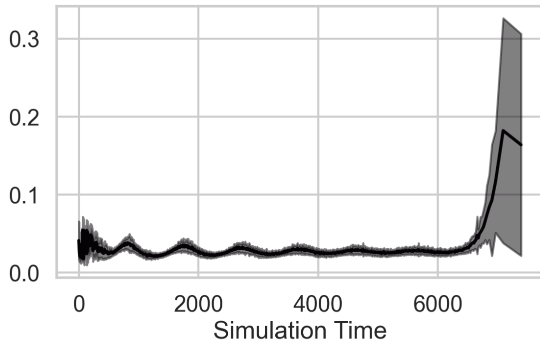


(g) Jm WIP to Arrival Time Ratio Behavior.

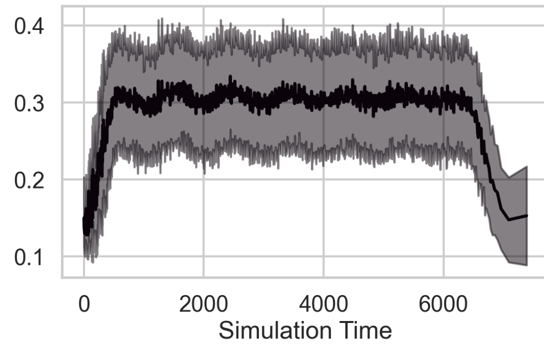


(h) Jm Work Completion Rate Behavior.

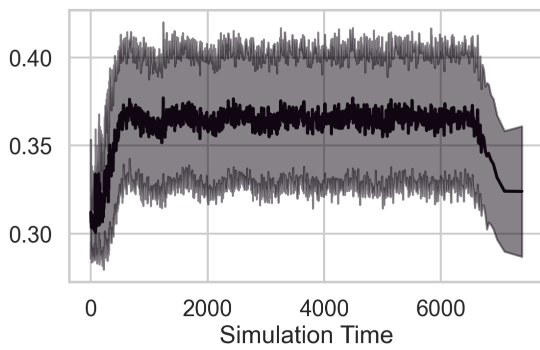
Figure C.6: Jm Feature Behavior (41-48).



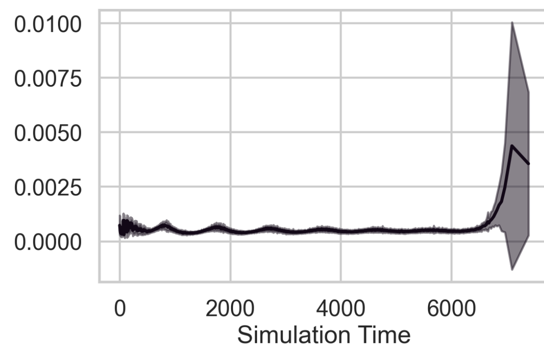
(a) FJc Buffer Length Ratio Behavior.



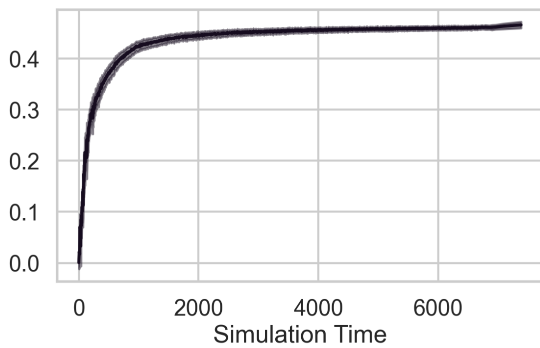
(b) FJc Buffer Load Average Behavior.



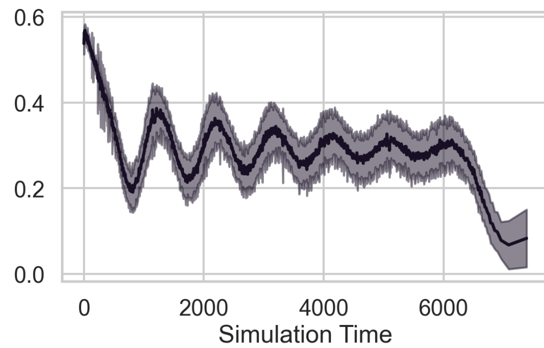
(c) FJc Buffer Load Standard Deviation Behavior.



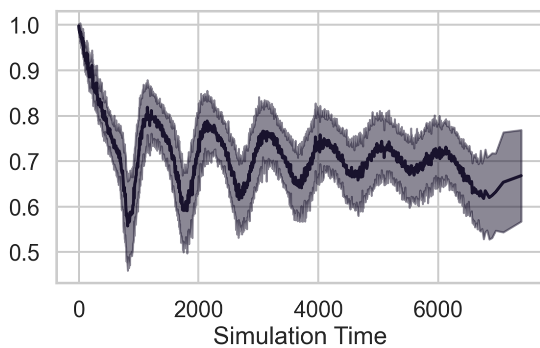
(d) FJc Buffer Time Ratio Behavior.



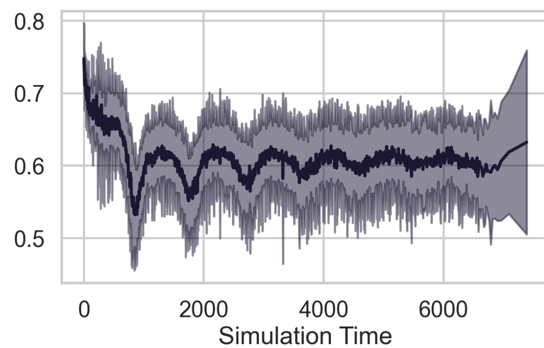
(e) FJc Decision Skip Ratio Behavior.



(f) FJc Normalized Duration Average Behavior.

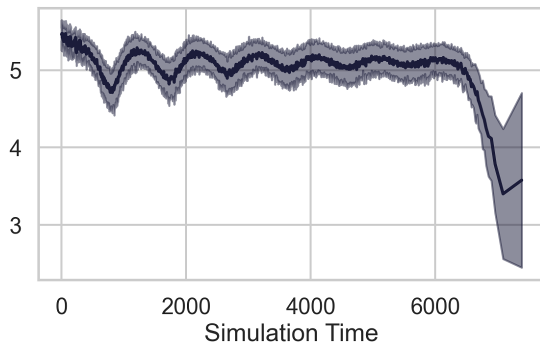


(g) FJc Duration Distance Mean Behavior.

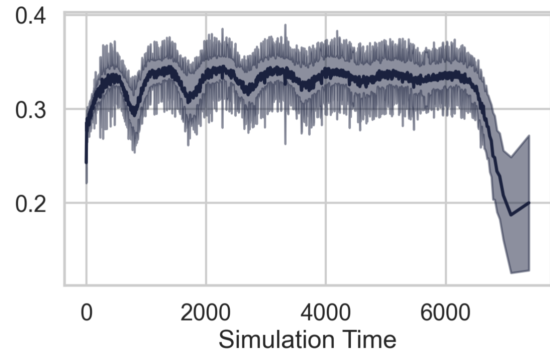


(h) FJc Duration Distance Standard Deviation Behavior.

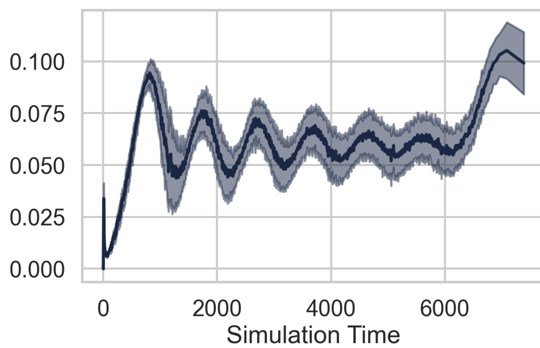
Figure C.7: FJc Feature Behavior (1-8).



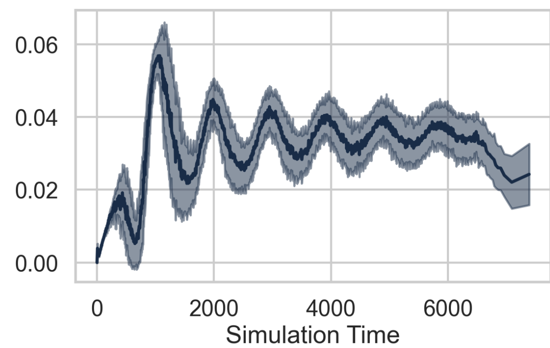
(a) FJc Duration Entropy Behavior.



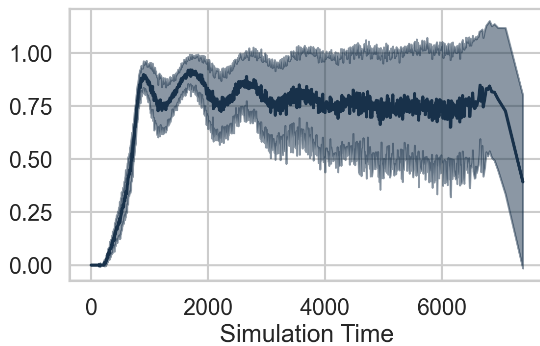
(b) FJc Normalized Duration Standard Deviation Behavior.



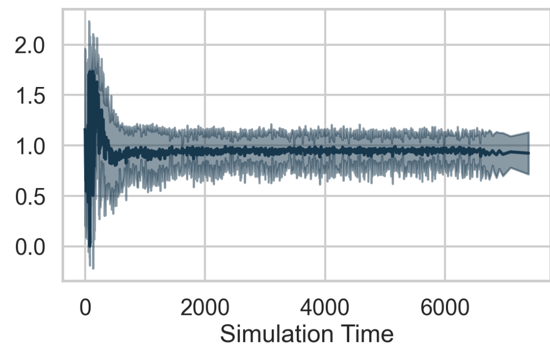
(c) FJc Estimated Flow Time Average Behavior.



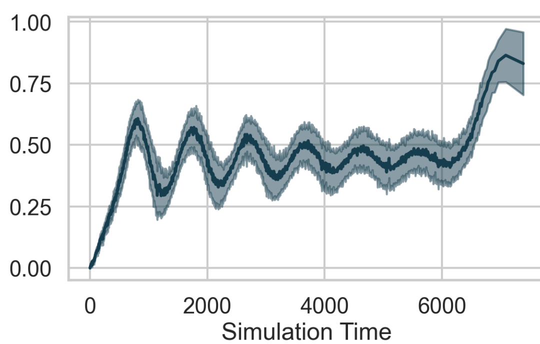
(d) FJc Estimated Flow Time Standard Deviation Behavior.



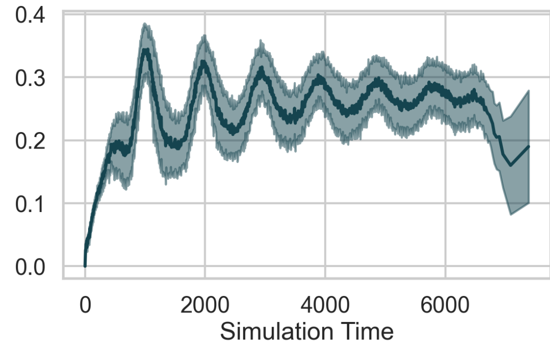
(e) FJc Estimated Tardiness Rate Behavior.



(f) FJc Heuristic Agreement Entropy Behavior.

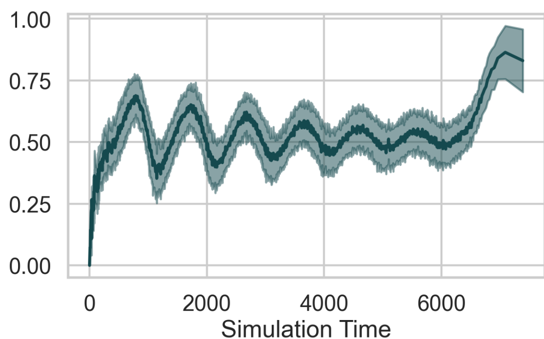


(g) FJc Job Operation Completion Rate Average Behavior.

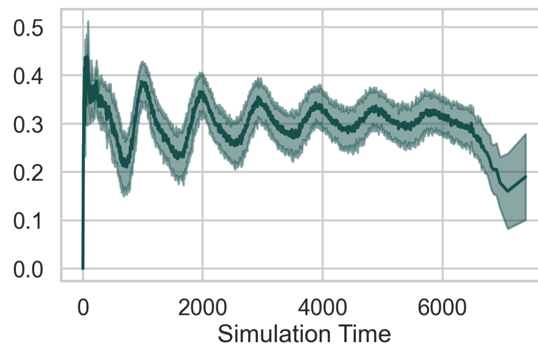


(h) FJc Job Operation Completion Rate Standard Deviation Behavior.

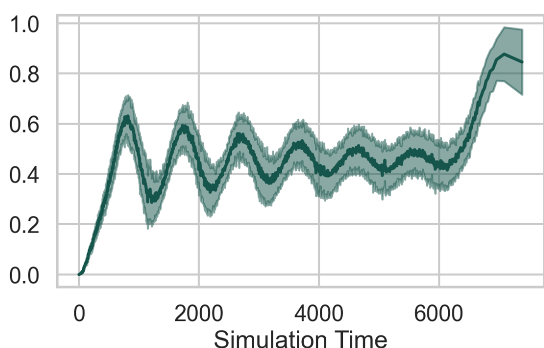
Figure C.8: FJc Feature Behavior (9-16).



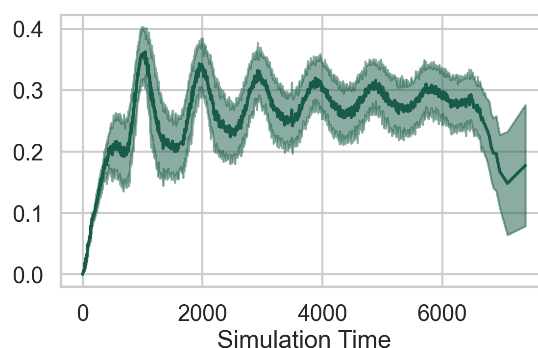
(a) FJc Job Operation Max Relative Completion Rate Average Behavior.



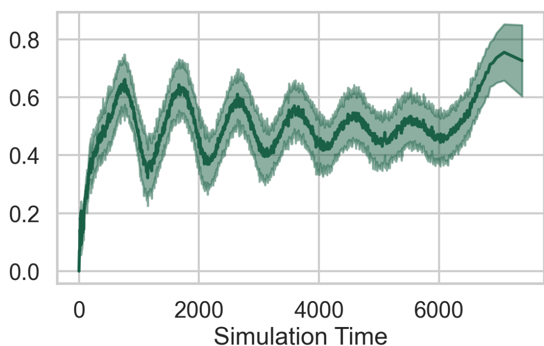
(b) FJc Job Operation Max Relative Completion Rate Standard Deviation Behavior.



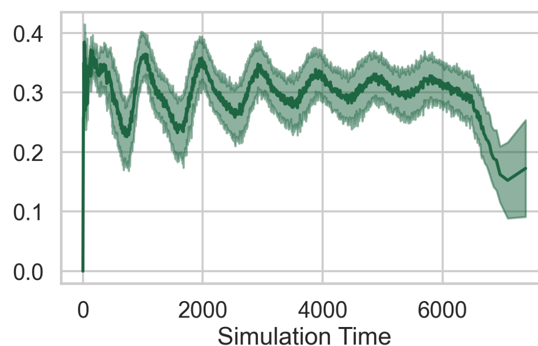
(c) FJc Job Work Completion Rate Average Behavior.



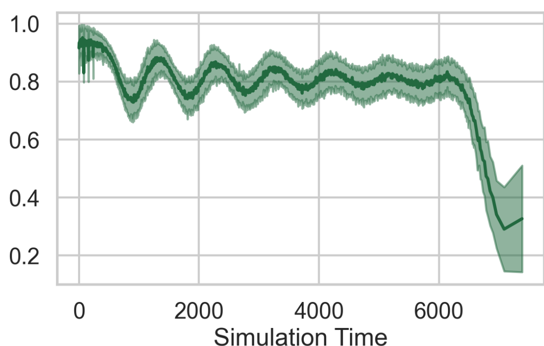
(d) FJc Job Work Completion Rate Standard Deviation Behavior.



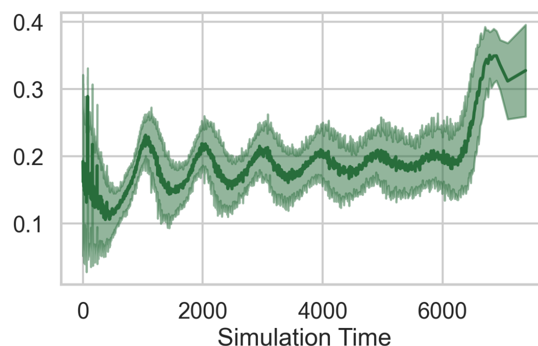
(e) FJc Max Relative Work Completion Rate Average Behavior.



(f) FJc Max Relative Work Completion Rate Standard Deviation Behavior.

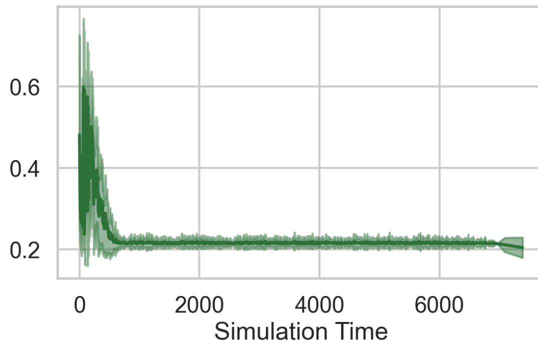


(g) FJc Kendall Tau Average Behavior.

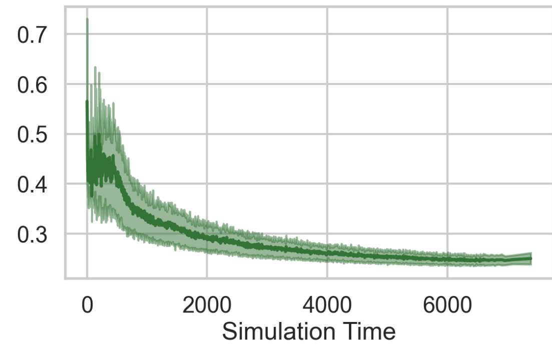


(h) FJc Kendall Tau Standard Deviation Behavior.

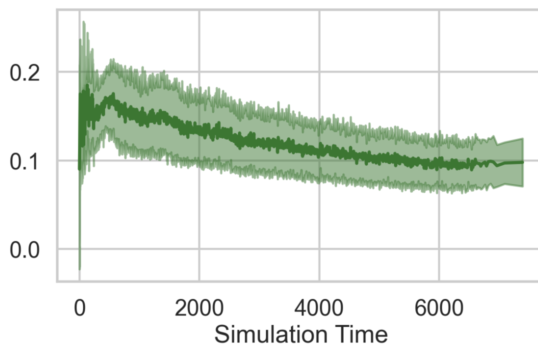
Figure C.9: FJc Feature Behavior (17-24).



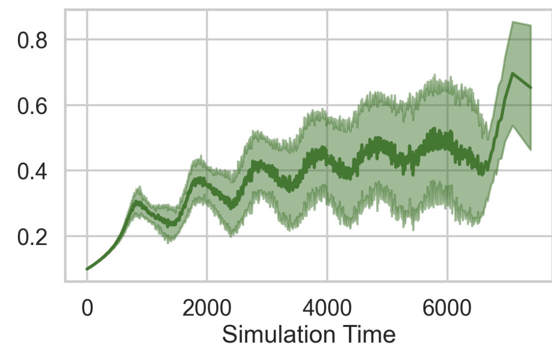
(a) FJc Legal Action to Job Ratio Behavior.



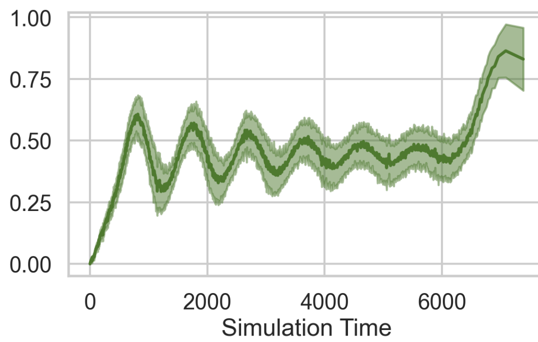
(b) FJc Legal Action Length Stream Average Behavior.



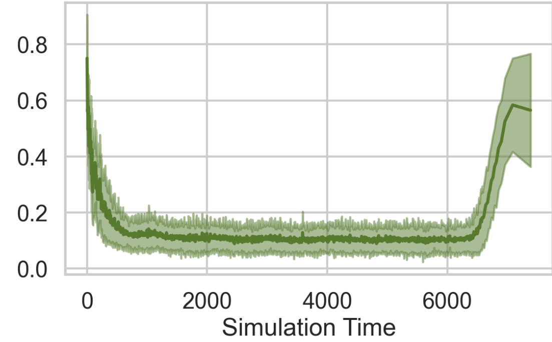
(c) FJc Legal Action Length Stream Standard Deviation Behavior.



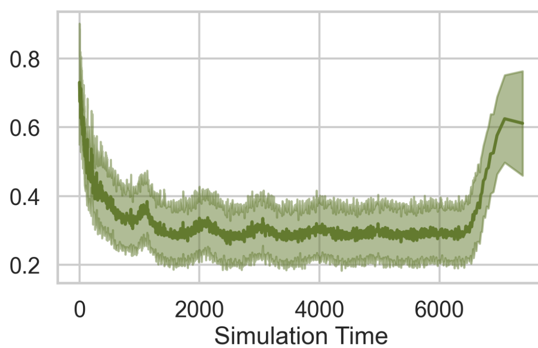
(d) FJc Makespan Lower Bound to Upper Bound Ratio Behavior.



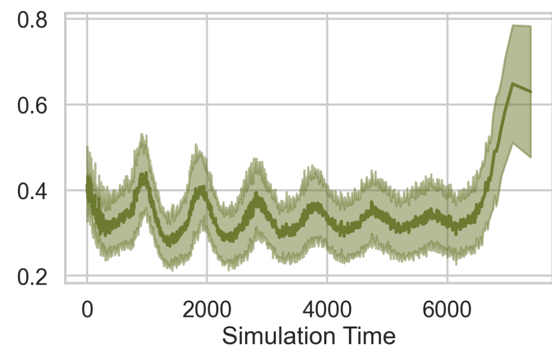
(e) FJc Operation Completion Rate Behavior.



(f) FJc Silhouette Maximum k Behavior.

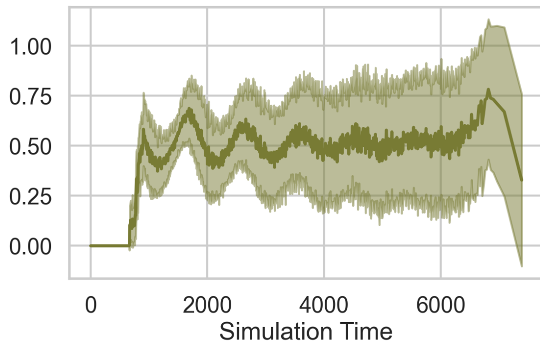


(g) FJc Silhouette Mid k Behavior.

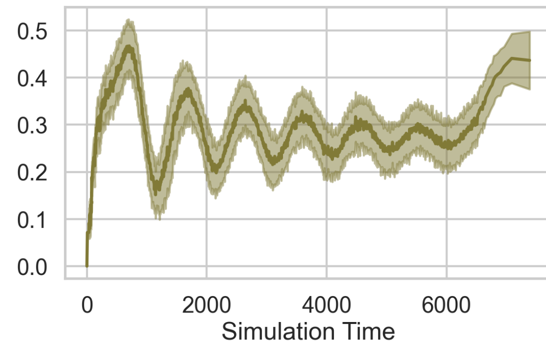


(h) FJc Silhouette Minimum k Behavior.

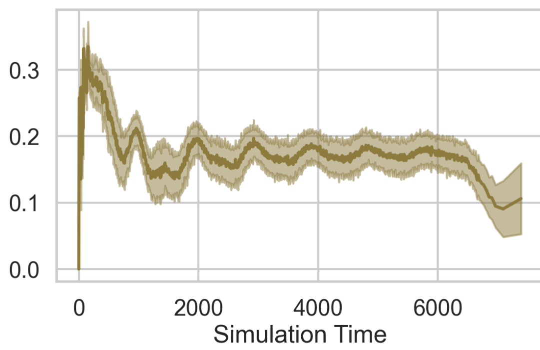
Figure C.10: FJc Feature Behavior (25-32).



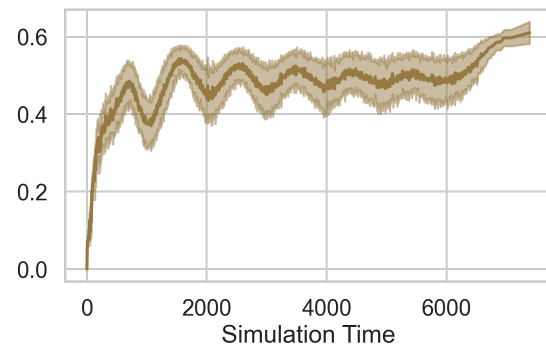
(a) FJc Estimated Tardiness Rate Behavior.



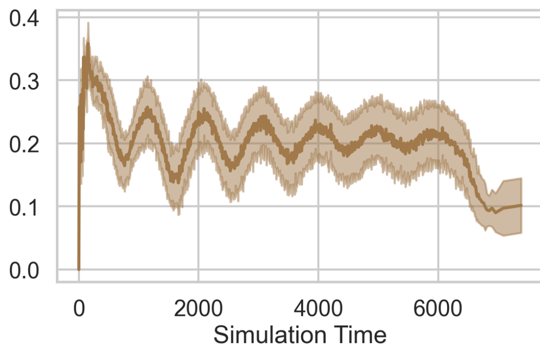
(b) FJc Absolute Job Throughput Time Average Behavior.



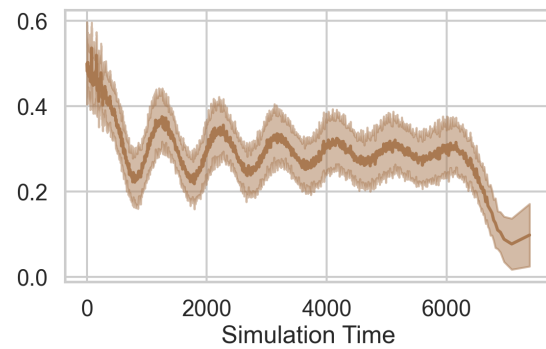
(c) FJc Absolute Job Throughput Time Standard Deviation Behavior.



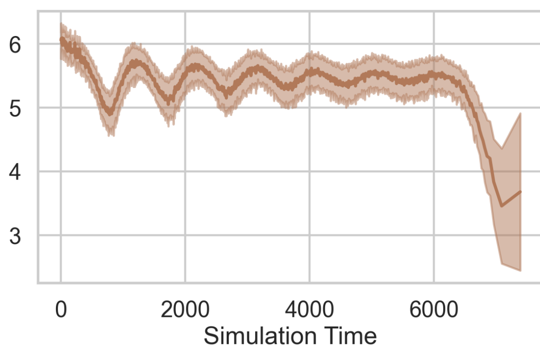
(d) FJc Relative Job Throughput Time Average Behavior.



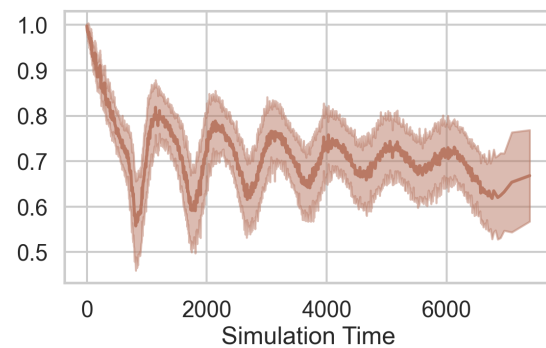
(e) FJc Relative Job Throughput Time Standard Deviation Behavior.



(f) FJc Normalized Type Average Behavior.

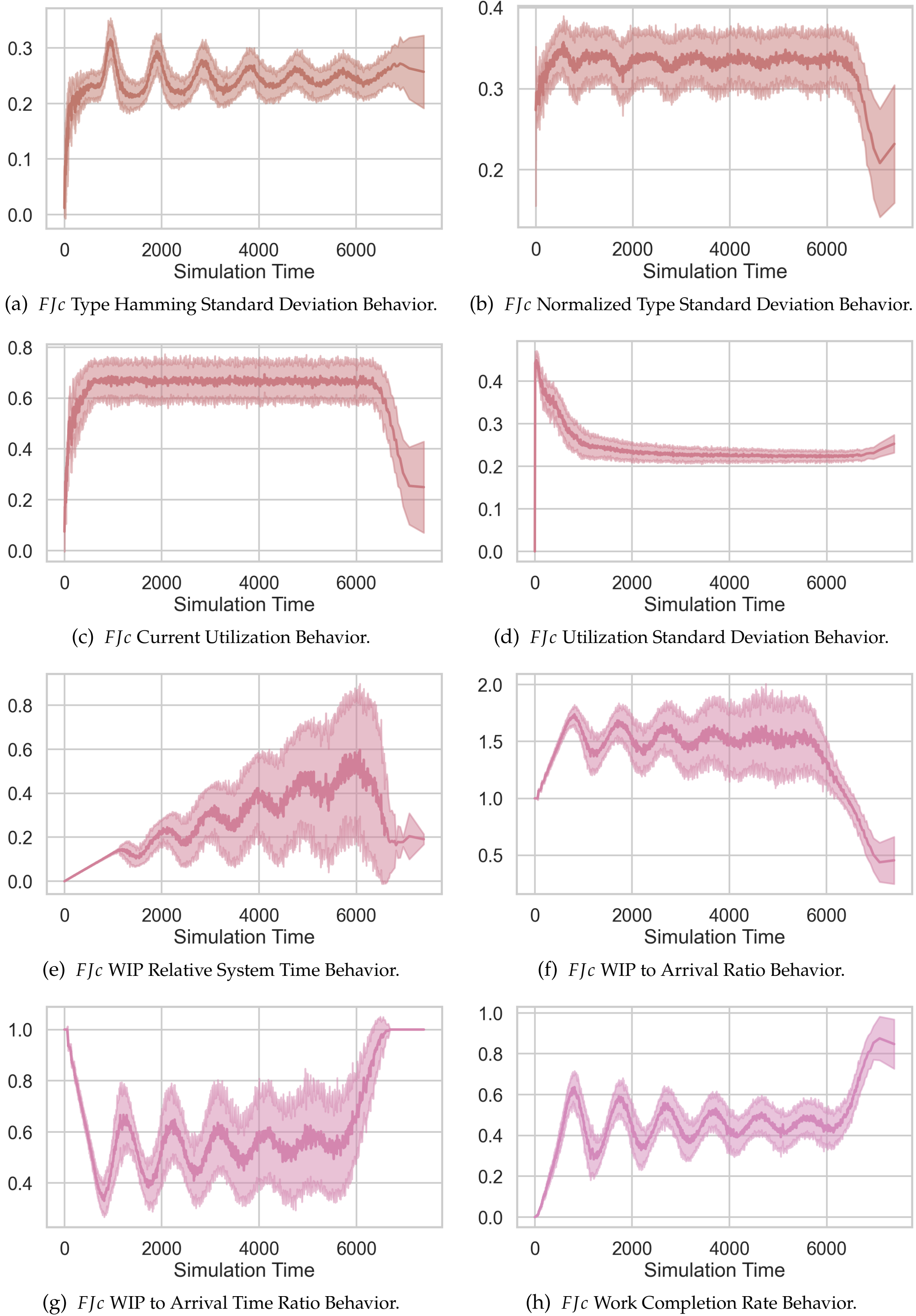


(g) FJc Type Entropy Behavior.



(h) FJc Type Hamming Mean Behavior.

Figure C.11: FJc Feature Behavior (33-40).



(a) FJc Type Hamming Standard Deviation Behavior.

(b) FJc Normalized Type Standard Deviation Behavior.

(c) FJc Current Utilization Behavior.

(d) FJc Utilization Standard Deviation Behavior.

(e) FJc WIP Relative System Time Behavior.

(f) FJc WIP to Arrival Ratio Behavior.

(g) FJc WIP to Arrival Time Ratio Behavior.

(h) FJc Work Completion Rate Behavior.

Figure C.12: FJc Feature Behavior (41-48).

Appendix D

Further Optimization Goals Measured During Experiments

Table D.1: Measured F_{ave} for Scheduling Algorithms (Controls) Optimizing C_{max}.

Algorithm	**FJc**	**Jm**
VBS	840.736 (0.000%)	810.460 (0.000%)
CP3	840.736 (-0.000%)	822.491 (-1.463%)
SimSearch	1267.921 (-33.692%)	843.656 (-3.935%)
MCTS	1272.316 (-33.921%)	839.429 (-3.451%)
AZ	1353.618 (-37.890%)	822.604 (-1.476%)
SPT	1354.452 (-37.928%)	822.176 (-1.425%)
LUDM	1365.916 (-38.449%)	837.129 (-3.186%)
SRPT	1367.517 (-38.521%)	836.281 (-3.088%)
EDD	1367.961 (-38.541%)	841.744 (-3.716%)
MTPO	1370.164 (-38.640%)	835.772 (-3.029%)
LOR	1373.317 (-38.781%)	848.197 (-4.449%)
DQN	1376.644 (-38.929%)	900.539 (-10.003%)
MOR	1388.150 (-39.435%)	901.893 (-10.138%)
LTPO	1396.319 (-39.789%)	919.581 (-11.866%)
LRPT	1398.457 (-39.881%)	927.345 (-12.604%)
LPT	1411.944 (-40.455%)	955.767 (-15.203%)
RND1	2214.957 (-62.043%)	863.536 (-6.146%)
RND2	2317.412 (-63.721%)	884.272 (-8.347%)

Table D.2: Measured T_{ave} for Scheduling Algorithms (Controls) Optimizing C_{max}.

Algorithm	**FJc**	**Jm**
VBS	100.377 (0.000%)	93.883 (0.000%)
CP3	100.377 (-0.000%)	105.460 (-10.978%)
SimSearch	439.192 (-77.145%)	124.543 (-24.618%)
MCTS	443.566 (-77.371%)	120.974 (-22.394%)
AZ	524.627 (-80.867%)	104.557 (-10.209%)
SPT	525.433 (-80.896%)	104.233 (-9.929%)
LUDM	536.667 (-81.296%)	112.163 (-16.298%)
SRPT	537.871 (-81.338%)	111.265 (-15.622%)
EDD	538.166 (-81.348%)	113.003 (-16.920%)
TPO	540.415 (-81.426%)	114.502 (-18.007%)
LOR	544.149 (-81.553%)	121.753 (-22.890%)
DQN	547.622 (-81.670%)	171.611 (-45.293%)
MOR	558.705 (-82.034%)	173.565 (-45.909%)
LTPO	567.311 (-82.307%)	190.628 (-50.751%)
LRPT	569.303 (-82.368%)	198.222 (-52.637%)
LPT	582.366 (-82.764%)	217.146 (-56.765%)
RND1	1384.345 (-92.749%)	137.717 (-31.829%)
RND2	1486.669 (-93.248%)	153.701 (-38.918%)

Table D.3: Measured Utl_{ave} for Scheduling Algorithms Optimizing C_{max}.

Algorithm	**FJc**	**Jm**
CP3	0.850 (+27.975%)	0.484 (+0.728%)
SimSearch	0.780 (+21.466%)	0.484 (+0.749%)
MCTS	0.775 (+21.007%)	0.484 (+0.739%)
AZ	0.765 (+19.963%)	0.484 (+0.578%)
SPT	0.765 (+19.905%)	0.484 (+0.601%)
LUDM	0.763 (+19.686%)	0.483 (+0.546%)
MTPO	0.763 (+19.684%)	0.484 (+0.553%)
DQN	0.762 (+19.652%)	0.484 (+0.584%)
MOR	0.762 (+19.651%)	0.484 (+0.703%)
SRPT	0.762 (+19.644%)	0.483 (+0.410%)
EDD	0.762 (+19.640%)	0.483 (+0.421%)
LOR	0.762 (+19.597%)	0.483 (+0.394%)
RPT	0.762 (+19.577%)	0.484 (+0.700%)
LTPO	0.761 (+19.503%)	0.484 (+0.577%)
LPT	0.759 (+19.277%)	0.483 (+0.481%)
RND1	0.631 (+2.874%)	0.484 (+0.557%)
RND2	0.618 (+0.810%)	0.483 (+0.523%)
VBS	0.613 (0.000%)	0.481 (0.000%)