Methodology for Self-Adaptively Solving Multi-Objective Scheduling Problems

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik der
Otto-von-Guericke-Universität Magdeburg

von **Abdulrahman Nahhas, M. Sc.**
geboren am 01.01.1990 in Aleppo

Gutachter:
Prof. Dr. Klaus Turowski
Prof. Dr. Steffen Straßburger
Prof. Dr. Gunter Saake

Magdeburg, den 24. Juni 2024

Abstract

Scheduling practices are critical decision-making processes that substantially influence the overall performance of cloud and manufacturing environments. Therefore, scheduling problems have been a primary concern of practitioners and scholars in this field for decades. The majority of scheduling problems are known NP-hard optimization problems. Hence, heuristic and improvement methods have been conventionally adopted to address scheduling concerns. Heuristic methods exhibit a light execution time but fail to sustain high solution quality for solving complex problems. Improvement methods deliver high-quality solutions but are associated with high computational effort.

To mitigate the individual limitations of both methods, scholars started investigating hybrid solution methods that may combine their advantages. The individual limitations of the conventional methods, in addition to the complex nature of the scheduling problem, result in a poor practical adoption of presented scheduling methods. Recently, Deep Reinforcement Learning (DRL) methods substantiated a fundamental breakthrough and have been successfully adopted in the gaming domain. The foundational design of DRL methods includes optimization elements, making them suitable for addressing scheduling problems.

Therefore, a scheduling methodology is presented that efficiently facilitates the combined utilization of heuristic, metaheuristic, and deep reinforcement learning methods to solve scheduling problems in cloud and manufacturing environments. Since most industrial scheduling problems are subject to multi-objective optimization measures, the methodology addresses scheduling concerns considering system efficiency and customer satisfaction objective measures. Parallelization and scalability technologies have been adopted to design and develop the presented artifact to achieve computational efficiency.

To conduct the research systematically, the proposed methodology relies on the Design Science Research (DSR) framework and adheres to its fundamental design activities. The identified research gap, validated by the theoretical findings and the needs of application environments, is translated into functional and non-functional requirements of the artifact. The derived functional and non-functional requirements are then mapped into functionality layers to define the overall functional structure of the proposed methodology. The artifact is designed using component and modular design practices to address single-stage and multi-stage scheduling problems in cloud and manufacturing, respectively.

The combined utilization of simulation, heuristic, improvement, and deep reinforcement learning methods was achieved by designing and developing a scheduling data model, several optimization encoding models for scheduling problems, DRL scheduling models, and a DRL evaluation model. The developed scheduling data model facilitates flexible instantiation of the methodology to address single-stage or multi-stage scheduling problems considering multiple objective measures. The subsequent implementation of the artifact design is presented as a proof of concept and evaluated based on the DRS framework. The developed prototype is designed to support a multi-architecture infrastructure deployment and execution. The simulation and heuristic, as well as the artifact's optimization and machine learning subsystems, are developed and deployed with parallelization and scalability features.

The developed prototype is evaluated on multiple use cases to address multi-objective scheduling problems in cloud and manufacturing environments. Its utility was evaluated for solving real multi-stage scheduling problems in manufacturing environments. Compared to related works, the artifact's optimization and DRL methods delivered, on average, 31.7% improved solutions in minimizing the system efficiency objective measures. The solutions also minimized penalties and delays by 33.3%, contributing to higher customer satisfaction.

Abstract in German

Planungsabläufe sind entscheidende Prozesse, die die Gesamtleistung von Cloud- und Produktionsumgebungen maßgeblich beeinflussen. Daher beschäftigen sich Praktiker und Wissenschaftler seit Jahrzehnten intensiv mit Planungsproblemen. Die meisten Maschinenplanungsprobleme gelten als NP-schwere Optimierungsprobleme, weshalb häufig Heuristiken und Optimierungsmethoden zur Lösung eingesetzt werden. Heuristische Methoden zeichnen sich durch kurze Ausführungszeiten aus, können jedoch bei komplexen Problemen keine hohe Lösungsqualität garantieren. Im Gegensatz dazu liefern Optimierungsmethoden hochwertige Lösungen, sind jedoch mit erheblichem Rechenaufwand verbunden.

Um die Nachteile beider Methoden zu überwinden, haben Wissenschaftler hybride Lösungsmethoden erforscht, die die Vorteile beider Ansätze kombinieren. Aufgrund der individuellen Einschränkungen herkömmlicher Methoden sowie der Komplexität von Maschinenplanungsproblemen finden diese Ansätze jedoch selten praktische Anwendung. In den letzten Jahren haben Methoden des Deep Reinforcement Learning (DRL) bedeutende Fortschritte erzielt und wurden erfolgreich im Gaming-Bereich eingesetzt. Das Grundkonzept von DRL-Methoden beinhaltet Optimierungselemente, was sie für die Lösung von Maschinenplanungsproblemen besonders geeignet macht.

Deshalb wird eine neue Scheduling-Methodik vorgestellt, die eine effiziente und präzise Kombination von heuristischen, metaheuristischen und Deep Reinforcement Learning-Methoden zur Lösung von Maschinenplanungsproblemen in Cloud- und Produktionsumgebungen ermöglicht. Da die meisten industriellen Maschinenplanungsprobleme multikriteriellen Optimierungsmaßnahmen unterliegen, berücksichtigt diese Methodik sowohl die Systemeffizienz als auch die Kundenzufriedenheit als Zielgrößen. Bei der Konzeption und Entwicklung des vorgestellten Artefakts wurden Technologien zur Parallelisierung und Skalierbarkeit genutzt, um eine hohe Recheneffizienz zu gewährleisten.

Um die Forschung systematisch durchzuführen, stützt sich die vorgeschlagene Methodik auf den Rahmen der Design Science Research (DSR) und folgt deren grundlegenden Designaktivitäten. Die identifizierte Forschungslücke, die durch theoretische Erkenntnisse und praktische Bedürfnisse validiert wurde, wird in funktionale und nicht-funktionale Anforderungen an das Artefakt übersetzt. Diese Anforderungen werden dann in verschiedene Funktionalitätsschichten abgebildet, um die gesamte funktionale Struktur der Methodik zu definieren. Das Artefakt wird unter Verwendung komponenten- und modularer Techniken entwickelt, um sowohl einstufige als auch mehrstufige Maschinenplanungsprobleme in Cloud- und Fertigungsumgebungen zu lösen.

Die integrierte Nutzung von Simulations-, Heuristik-, Verbesserungs- und Deep Reinforcement Learning-Methoden wurde durch den Entwurf und die Entwicklung eines Datenmodells, mehrerer Optimierungskodierungsmodelle für Maschinenplanungsprobleme, DRL-modelle und eines DRL-Evaluierungsmodells erreicht. Die anschließende Implementierung des Artefaktdesigns wird als Proof of Concept vorgestellt und auf der Grundlage des DRS-Frameworks evaluiert. Der entwickelte Prototyp ist so konzipiert, dass er den Einsatz und die Ausführung einer Multi-Architektur-Infrastruktur unterstützt. Die Simulation und Heuristik sowie die Teilsysteme für Optimierung und maschinelles Lernen des Artefakts werden mit Parallelisierungs- und Skalierungsfunktionen entwickelt und eingesetzt.

Der entwickelte Prototyp wurde anhand mehrerer Anwendungsfälle evaluiert, um multikriterielle Maschinenplanungsprobleme in Cloud- und Produktionsumgebungen zu lösen. Sein Nutzen wurde für die Lösung realer mehrstufiger Maschinenplanungsprobleme in Produktionsumgebungen evaluiert. Im Vergleich zu verwandten Arbeiten lieferten die Optimierungs- und DRL-Methoden des Artefakts im Durchschnitt $31,7\%$ bessere Lösungen bei der Minimierung der Makepan und der Gesamtzahl der Hauptrüstzeiten, was zu einer höheren Systemeffizienz beitrug. Die Lösungen minimierten auch Penalties und Lieferterminverzögerungen um $33,3\%$, was zu einer höheren Kundenzufriedenheit beitrug.

Contents

List of Figures

List of Tables

List of Abbreviations

A3C ..... Asynchronous Advantage Actor-Critic

ACO ..... Ant Colony Optimization

AHP ..... Analytical Hierarchy Process

AMQP ..... Advanced Message Queuing Protocol

ANN ..... Artificial Neural Network

AOI ..... Automated-Optical-Inspection

ARIMA ..... Autoregressive Integrated Moving Average model

ARM64 ..... 64-bit Advanced RISC Machin

BFD ..... Best-Fit-Decreasing

CC ..... Conformal Coating

CEM ..... Cross Entropy Method

CH ..... Critical Horizon

CICS ..... Complex-Instruction-Set Computers

CMA ..... Covariance Matrix Adaptation

CPS ..... Cyber-Physical Systems

CPU ..... Central Processing Unit

CRI ..... Container Runtime Interface

DA-FFI ..... Deadline-Aware Family-Fit-Increasing

DA-SI ..... Deadline-Aware Sequencing-Increasing

DDPG ..... Deep Deterministic Policy Gradient

DE ..... Differential Evolutionary

DEAP ..... Distributed Evolutionary Algorithms in Python

DES ..... Discrete Event Simulation

DL ..... Deep Learning

DNN ..... Deep Neural Network

DQN ..... Deep Q-Network

DRL ..... Deep Reinforcement Learning

DSR ..... Design Science Research

DWA-FFD ..... Energy-Aware Family-Fit-Decreasing

DWA-FFI ..... Deadline-Workload-Aware Family-Fit-Increasing

EA-FFD ..... Energy-Aware Family-Fit-Decreasing

EC ..... Execution Costs

EDD ..... Earliest Due Date

EDD-FLPT ..... EDD-Family Longest-Processing-Time

EDD-FSPT ..... EDD-Family Shortest Processing Time

FD-WI ..... Family-Decreasing Workload-Increasing

FFD ..... First-Fit-Decreasing

FFI ..... First Fit Increasing

FI-WI ..... Family-Increasing Workload-Increasing

FIFO ..... First-In-First-Out

GA ..... Genetic Algorithm

GGA ..... Grouping Genetic Algorithm

gRPC ..... General-purpose Remote Procedure Call

GWO ..... Grey Wolf Optimization

HFS ..... Hybrid Flow Shop

HHO ..... Harris Hawks Optimizer

HL ..... Heuristic Libaray components

IaaS ..... Infrastructure as a service

IL ..... Imitation Learning

IQR ..... Interquartile Range

IS ..... Information System

ISA ..... Instruction-Set-Architecture

ISO ..... International Organization for Standardization

IT ..... Information Technology

JDK ..... Java Development Kit

JSON ..... JavaScript Object Notation

KL-Divergence ..... Kullback-Leibler Divergence

LIFO ..... Last-In-First-Out

LPT ..... Longest Processing Time

LRPT ..... Longest Remaining Processing Time

MARWIL ..... Monotonic Advantage Re-Weighted Imitation Learning

MBFD ..... Modified Best-Fit Decreasing

MC ..... Maximum Children

MDP ..... Markov Decision Process

MFED ..... Medium-Fit Power Efficient Decreasing

MST ..... Minimum Slack Time

NEAT ..... NeuroEvolution of Augmenting Topologies

NOP ..... Number of Operations

NSGA ..... Nondominated Sorting Genetic Algorithm

NSGA III ..... Non-Dominated Sorting Genetic Algorithm three

OBL ..... Opposition-Based Learning

PABFD ..... Power Aware Best Fit Decreasing

PCB ..... Printed Circuit Board

PCP ..... Partial Critical Path

PDRs ..... Priority Dispatching Rules

PEBFD ..... Power Efficient Best-Fit Decreasing

PEFFD ..... Power Efficient First-Fit Decreasing

PI ..... Problem Instance

PM ..... Parallel Machine

PPO ..... Proximal Policy Optimization

PSF ..... Pre-Subcontractor First

PSO ..... Particle Swarm Optimization

QoS ..... Quality of Service

RC ..... Ratio of Criticality

RISC ..... Reduced-Instruction-Set Computers

RL ..... Reinforcement Learning

RPC ..... Remote Procedure Call

SA ..... Simulated Annealing

SaaS ..... Software as a Service

SD ..... System Dynamics

SLA ..... Service Level Agreements

SLR ..... Systematic Literature Review

SMD ..... Surface Mounting Device

SP ..... Scheduling Problem

SPT ..... Shortest Processing Time

SRPT ..... Shortest Remaining Processing Time

SS ..... Selective Soldering

SST ..... Shortest Setup Time

TD3 ..... Twin Delayed Deep Deterministic Policy Gradient

TDE ..... Temporal Difference Error

TRETA ..... Total Resource Execution Time Aware Algorithm

TRPO ..... Trust Region Policy Optimization

TS ..... Tabu Search

UI ..... User Interface

UML ..... Unified Modeling Language

VM ..... Virtual Machine

WA-FFD ..... Workload-Aware Family-Fit-Decreasing

WFD ..... Worst-Fit-Decreasing

YAML ..... Yet Another Markup Language

1 Introduction

In this chapter, we present the motivation of the dissertation, which systematically discusses the relevant research foundations and application environments to highlight the challenges. We express the identified challenges and possible opportunities in the form of research hypotheses, which this thesis aims to validate. Based on the research hypotheses, we articulate the overall research objective of the dissertation. We thoroughly discuss the adapted research methodology and highlight the research questions to accomplish the research objective. Following, we provide an overview of the author's main published works and communicate their contribution to the intended artifact of this research. Finally, we present the outlines of the thesis based on the adopted methodology.

1.1 Motivation

In Information Technology (IT) and advanced manufacturing, the introduction and establishment of certain cloud and Industry 4.0 technologies have fundamentally changed business needs. For instance, efficient and accurate decision-making processes become necessary to sustain competitive advantages in different markets (Pinedo, 2012, p. 1). A crucial subset of these decision-making processes enforces scheduling and sequencing practices in various cloud and manufacturing environments (Pinedo, 2012, p. 3; Baker and Trietsch, 2009, p. 4). Scheduling is the process of allocating limited resources to complete a set of tasks in some technological order (Pinedo, 2012, p. 1). Scheduling practices are present and crucial in every service or manufacturing sector (Pinedo, 2012, pp. 1-6), such as scheduling patients in some healthcare clinics (Nahhas et al., 2017b), scheduling orders or jobs to be processed in a manufacturing environment (Nahhas et al., 2017a; Pinedo, 2012, p. 5), or scheduling jobs for processing in cloud environments (Nahhas et al., 2021a; Jiang et al., 2020). For four decades, scholars and practitioners have been intensively investigating different variations of scheduling problems due to their essential role in the daily operations of these industries (Pinedo, 2012, p. 537; Baker and Trietsch, 2009, p. 2).

Depending on a considered system, scheduling activities can be reduced to single-stage or multi-stage scheduling problems. In cloud environments, scheduling activities are overwhelmingly formulated as single-stage scheduling problems (Arunarani et al., 2019, p. 413; Pires and Barán, 2015, p. 165), while few research deals with multi-stage or workflow scheduling in cloud environments (Arunarani et al., 2019, p. 413). In a single-stage scheduling environment, a set of m number of physical machines available in parallel to process or host a set of application instances, virtual machines, or jobs. In manufacturing

environments, scheduling activities are significantly more complex since jobs may require processing in multiple production stages, in which machines are available to process them in parallel. In terms of problem formulation, multi-stage scheduling in manufacturing is almost identical to workflow scheduling in cloud environments. However, they are addressed by considering slightly different objective measures.

In operations research, these scheduling problems are referred to as Hybrid Flow Shop (HFS) scheduling problems (Neufeld et al., 2023, p. 1). HFS scheduling problems represent a wide range of industrial manufacturing systems since they formally express multi-stage assembly production systems (Pinedo, 2012, p. 151; Ruiz and Vázquez-Rodríguez, 2010, p. 5; Ribas et al., 2010, p. 1449). An HFS production environment consists of S processing stages in series. Each processing stage offers m parallel machines. Usually, the operations of such systems are subject to technological constraints expressed in specific processing orders of jobs. Each job J_j has to be processed in each processing stage by one of the available machines (Pinedo, 2012, p. 15). For decades, scheduling theory mediated the transfer of theoretical scheduling concepts to industrial practices. However, presented mathematical models and formulation of scheduling problems in scheduling theory are dominated by NP-hard problems (Challita et al., 2017, p. 345; Neufeld et al., 2016, p. 58; Pinedo, 2012, pp. 26-28; Gupta and Stafford, 2006, p. 703; Koulamas, 1994, p. 1036; Lenstra et al., 1977, pp. 344-349; Lenstra et al., 1977, pp. 117-123). *Consequently, traditional analytical solutions are too computationally expensive* and often cannot be applied to find optimal or sub-optimal solutions.

Therefore, conventional solution techniques for solving industrial scheduling problems are overwhelmingly dominated by the adoption of heuristic and improvement methods (Neufeld et al., 2023, pp. 4-7; Arunarani et al., 2019, p. 408; Challita et al., 2017, p. 348; Neufeld et al., 2016, p. 63; Pires and Barán, 2015, p. 9). A constructive heuristic is usually designed to construct a solution with no further improvement mechanism. In practice, constructive heuristics are developed using simple procedural logic that often relies on some priority index describing job characteristics. Priority Dispatching Rules (PDRs) are well-established constructive heuristics often used in different industrial contexts to deal with scheduling problems (Oukil and El-Bouri, 2021, p. 389; Rolf et al., 2020a, p. 443; Baker and Trietsch, 2009, pp. 58-59; Blackstone et al., 1982, pp. 27-29). The Earliest Due Date (EDD) is a PDR, which ranks jobs according to their due date. It is often applied to schedule jobs in a manufacturing environment (Rolf et al., 2020b, p. 1588; Blackstone et al., 1982, p. 29) or processing in a cloud environment (Menaka and Sendhil Kumar, 2022, pp. 2-3; Moges and Abebe, 2019, p. 4; Murad et al., 2024, p. 233). The simple design of these heuristics makes them very attractive since they do not require expert engineering skills to be developed (Pinedo, 2012, pp. 376-377; Ross, 2005, pp. 530-531). *However, constructive heuristics lack performance and quality of solution if business needs necessitate optimizing multiple objective concerns* (Pinedo, 2012, p. 377).

Therefore, improvement methods are usually applied to solve multi-objective scheduling problems (Kruekaew and Kimpan, 2022; Nahhas et al., 2021a,b; Ross, 2005). Typically, improvement heuristics are based on combining simple constructive heuristics to construct

an initial solution and improvement mechanism, which is employed to enhance the quality of the initial solution iteratively. Metaheuristics are a subset of improvement techniques, which are often inspired by different natural concepts such as evolution theory in the case of Genetic Algorithms (GA) presented by Holland (1975), the behavior of swarm in the case of Particle Swarm Optimization (PSO) (Liao et al., 2007), or annealing of metal in the case of Simulated Annealing (SA) proposed by Kirkpatrick et al. (1983). The improvement mechanism of the metaheuristic technique is designed to systematically revise the initial solution or population of solutions to achieve improvement in terms of minimizing or maximizing some objective values (Ross, 2005, p. 530). They are often used to address multi-objective scheduling problems and can deliver high-quality solutions (Neufeld et al., 2023, pp. 4-7; Pires and Barán, 2015, pp. 164; Ross, 2005, pp. 530-531). However, *improvement methods are usually subject to high computational effort for achieving superior solutions* (Ross, 2005, p. 531).

The computational effort limitation of improvement methods motivated scholars and practitioners to pursue more efficient techniques that deliver acceptable solutions using hybrid solution methods (Bhattacharyya, 2018; Dey et al., 2018). A hybrid solution technique combines any two different types of solution techniques for solving a given problem. Although the term is relatively recent, the notion has been used for decades to describe algorithmic solutions that inherit a mix of any kind as presented in (Crowston et al., 1963). Many contributions can be found in the literature combining heuristic and metaheuristic techniques for addressing scheduling problems (Remesh et al., 2023; Rashida et al., 2020; Rolf et al., 2020a; Nahhas et al., 2017a; Amoretti et al., 2013; Ross, 2005). *Although hybrid methods maintain an acceptable execution time compared to improvement methods for solving a given problem, they are still vulnerable to modification in a considered system.* As a result, they might lack robustness and adaptivity to constantly adopting new technologies.

1.2 Research gap

In addition to individual limitations of previously discussed solution methods, the majority of them share a common adoption drawback. Despite the extensive research efforts invested in the field of scheduling, a lack of industrial adoption of these solution methods is evident in various application fields (Romero-Silva et al., 2022, p. 4; Ross, 2005, pp. 530-531; Reisman et al., 1997; Maccarthy and Liu, 1993). Reisman et al. (1997) reviewed forty years of research effort in scheduling and raised concerns that point to *oversimplification of real-world scheduling problems, which leads to neglecting the multi-objective and dynamic nature of scheduling in practice.* Furthermore, the authors stressed that 3 % of found methods can be deemed applicable for addressing scheduling problems in practice. Although the findings of Reisman et al. (1997) are quite surprising, the reasoning behind his findings is not. Already in the early nineties, Maccarthy and Liu (1993) discussed that ideal problem setup and abstraction practices in scheduling theory are far away from normal practical environments (Maccarthy and Liu, 1993, p. 70). Middle of two thou-

sand, Ross (2005) opened the same discussion and stressed the high development costs associated with problem-specific solution methods Ross (2005, pp. 530-531), that lack flexibility. Finally, two years ago Romero-Silva et al. (2022, p. 4) revisited the discussion and reiterated similar concerns by systematically analyzing the theory-practice gap in the scheduling field.

Eventually, the combinatorial nature of scheduling problems from a research perspective and the realistic application of researched methods in practice is an ever-to-be-addressed dilemma. Many scholars emphasized this gap and encouraged authors to address some of these limitations (Swan et al., 2022, p. 400; Romero-Silva et al., 2022, p. 4; Urquhart et al., 2019, p. 1345; Neufeld et al., 2016, p. 70; Ruiz and Vázquez-Rodríguez, 2010, p. 21; Reisman et al., 1997, p. 326). Therefore, this dissertation presents a scheduling methodology that combines the utilization of simulation, heuristic, metaheuristic, and Deep Reinforcement Learning (DRL) methods to address multi-objective scheduling problems. The objective is to leverage their combined potential and mitigate some of their limitations through their combined use.

Simulation methods are powerful techniques for modeling, simulating, and partially constructing solutions for complex optimization problems (Chica et al., 2020, pp. 324-325). In contrast to all discussed methods, adopting DRL methods for solving scheduling problems is poorly researched and may contribute to achieving adaptivity in solving scheduling problems. DRL methods inherit optimization character by design, which is based on the concept of Markov Decision Process (MDP) (Papadimitriou and Tsitsiklis, 1987a). A DRL algorithm, referred to as an agent, interacts with the environment according to a well-defined mechanism called the action space. The environment is typically an abstract model of the real system. Given the action space, the agent selects an action that is applied in the environment. Based on the quality of the suggested action, the agent earns a reward, which it seeks to maximize. Iteratively, the agent learns to optimize its behavior and enhance interaction with the environment.

In addition to integrating methodologically different solutions methods in the presented artifact, the notion of learning from solutions to scheduling problems is very promising from academic and industrial perspectives. Therefore, we aim to harmonize the integration of these methods to address realistic multi-objective scheduling problems. Incorporating expert experiences to solve complex scheduling problems using DRL methods may yield an adaptive scheduling methodology that leverages conventional methods' potential and DRL methods' adaptivity. The research and development efforts required to develop the pursued methodology are the core subject matters of this dissertation. The next section presents the design and objective of the thesis at hand.

1.3 Research design and objectives

We briefly discussed the lack of adaptivity of existing scheduling techniques in the literature (Romero-Silva et al., 2022) and stressed the importance of addressing multi-objective optimality concerns (Neufeld et al., 2023; Pires and Barán, 2015). Ideally, modern schedul-

ing techniques must deliver high-quality solutions with acceptable computational effort (Ross, 2005). Designing such a solution technique is subject to research and development challenges, which we intend to point out in this section. We express these challenges in the form of research hypotheses and articulate the overall research objective of the thesis. To meet the research objective, we formulate research questions that the thesis focuses on answering. The main research objective of the presented thesis relies on three research hypotheses, which we systematically investigate in the course of the thesis to develop the research artifact. Investigating the adoption of DRL methods promises to address complex scheduling problems if DRL methods can be trained to learn scheduling solutions. Therefore, we express the first hypothesis as follows:

Hypothesis 1
Learning from solutions to scheduling problems in cloud and manufacturing environments could be a significant step in instantly supporting scheduling decision-making processes. Therefore, deep reinforcement learning methods might be adopted to learn from solutions to scheduling problems.

In this hypothesis, we presume that DRL methods can be trained using solutions to scheduling problems. After sufficient training, they can be used to solve scheduling problems in cloud and manufacturing environments. The fact that DRL methods inherit an optimization nature motivates this assumption (Papadimitriou and Tsitsiklis, 1987a). Learning from solutions to scheduling problems or imitating expert behavior requires integrating multiple solution methods. Our previous discussion highlighted the key advantages and disadvantages of various solution methods often adopted to solve scheduling problems. It is crucial to establish a profound understanding of how we can integrate these techniques to develop a hybrid one that leverages their combined advantages. Therefore, the second hypothesis results from the first one and can be expressed as follows:

Hypothesis 2
Achieving an adaptive scheduling mechanism requires leveraging the light computational efforts of *heuristic methods*, robustness and high-quality solutions of *metaheuristic methods*, and the adaptive capabilities of *deep reinforcement learning methods*. Therefore, scheduling problems in cloud and manufacturing environments can be addressed using a methodology that integrates the utilization of heuristic, metaheuristic, and machine learning methods. This combination may adapt to the dynamic nature of these environments.

Scheduling problems in cloud and manufacturing environments are subject to various operational constraints and objective values. The majority of conventional optimization methods are usually designed or adopted to address single-objective scheduling problems (Neufeld et al., 2016; Pires and Barán, 2015). Furthermore, authors often argue that no single algorithm addresses all objective concerns at all times (Ghafari et al., 2022; Nahhas et al., 2019a; Ross, 2005). Therefore, employing multiple heuristics combined

with a metaheuristic method may yield better performance in solving multi-objective scheduling problems. To this extent, based on the first and second hypotheses, the third hypothesis discusses the prospect of utilizing several algorithms during a scheduling period, as presented here:

Hypothesis 3
Scheduling problems are inherently complex due to the dynamic nature of real cloud and manufacturing systems. Therefore, employing multiple heuristic methods controlled by a metaheuristic method during a scheduling period may yield better results than using them independently. The quality of the solution is subject to multiple objective measures that consider system efficiency and customer satisfaction.

In the course of this work, we will examine the previous hypotheses to seek their validation. Validating the third hypothesis may define how we should efficiently employ various heuristics during a scheduling period, which may deliver better results for addressing multi-objective optimization concerns. This, in turn, would require utilizing a metaheuristic method as a controlling or guiding mechanism to achieve high-quality solutions. Efficient integration of these methods lays out a road map to integrate them with deep reinforcement learning methods to achieve adaptivity of the pursued scheduling method and validate the second hypothesis. Finally, learning from solutions to scheduling problems confirms the first and foremost research hypothesis to achieve the overall research objective. Hence, we may express the overall research objective of the thesis at hand as follows.

Research objective
The research goal of the presented thesis is to present a scheduling methodology that facilitates efficient and accurate combined utilization of heuristic, metaheuristic, and deep reinforcement learning methods to solve scheduling problems in cloud and manufacturing environments. We strive to achieve the research objective by conceptualizing, designing, and developing a scheduling methodology that rapidly deploys integrated simulation, optimization, and deep reinforcement learning methods to solve scheduling problems. To achieve computational efficiency, parallelization and scalability technologies must be adopted. The majority of industrial scheduling problems are subject to multi-objective optimization measures. Therefore, the intended methodology must address scheduling concerns considering system efficiency and customer satisfaction objective measures.

One must rely on an appropriate scientific methodology to conduct research and development projects with a clear definition of scientific requirements and objectives. The nature of conducted research in information systems and comparable disciplines is usually characterized as behavioral or constructional (von Hevner et al., 2004). The research at hand exhibits a strong constructional nature. Based on the presented study by von Hevner et al. (2004), the Design Science Research (DSR) framework is suitable for investigating, designing, developing, and evaluating IT research artifacts that reflect actual

utility in their perspective environments. Therefore, the research is conducted following the DSR framework (von Hevner et al., 2004). Based on the DSR framework presented by von Hevner et al. (2004), the research artifact is presented in Figure 1.1. The intended research artifact is a flexible, scalable, and distributed scheduling methodology that integrates the utilization simulation, heuristic, metaheuristic, and deep reinforcement learning methods for solving multi-stage scheduling problems considering multi-objective optimality concerns.

The adapted research methodology is founded on three core pillars: The application environment, the Information Systems (IS) research artifact, and the Knowledge base (von Hevner et al., 2004). In contrast to (Orlikowski and Barley, 2001), the research artifact is positioned by von Hevner et al. (2004) as a broad *"core subject matter"*, which includes instantiations, constructs, models, or methods. To design the research artifact, we rely on several research foundations from the knowledge base due to the interdisciplinary nature of scheduling problems and their solution techniques, as presented in Figure 1.1.

Among others, we first investigate challenges in cloud operation management and resource planning in just-in-time manufacturing with a primary focus on scheduling problems to determine relevant requirements that must be addressed. Furthermore, we analyze and investigate the performance of formal heuristic and metaheuristic methods to design an efficient scheduling methodology. To integrate DRL methods, we explore the potential of the DRL technique for addressing scheduling concerns. We present a proof-of-concept following algorithm engineering practices to evaluate the performance of the scheduling methodology and rely on simulation techniques to develop evaluation and solution construction components. We further explore utilizing various open-source and cloud technologies to implement the prototype, considering flexibility, scalability, parallelization, and efficiency to address scheduling problems.

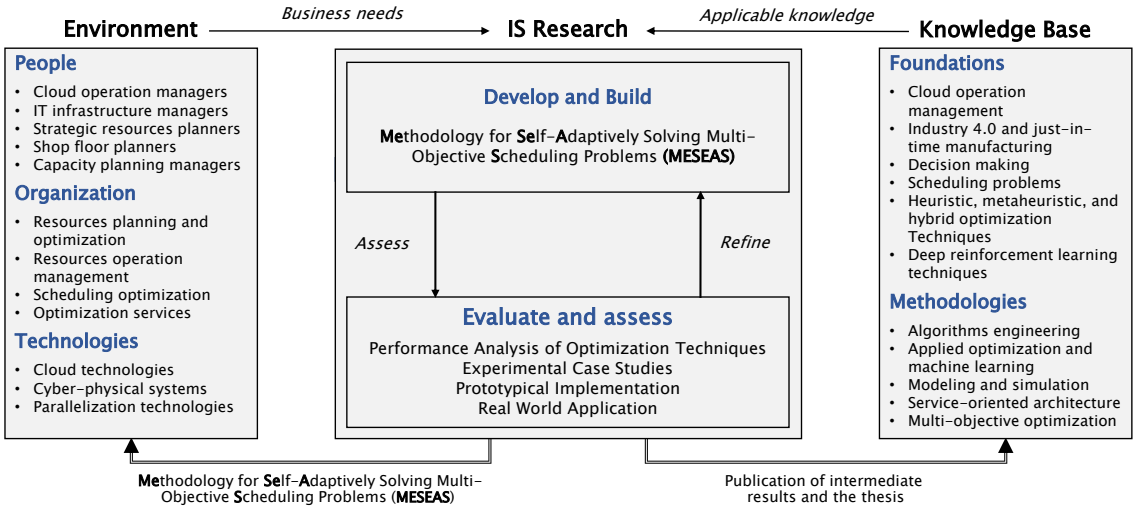


Figure 1.1: Adopted design science research methodology to conduct this research.

Every research artifact and its associated proof-of-concept is, in essence, an experiment that answers open question/s in the respective environment (von Hevner et al., 2004) based on (Newell and Simon, 1975). The epistemology of design science can be expressed in three phases forming three distinct levels of research: Conceptual, descriptive, and prescriptive research (Iivari, 2007). The descriptive research phase includes investigating and validating various hypotheses (Iivari, 2007) to derive grounded observations and empirical results (Daase et al., 2024). These results are used to conduct prescriptive research by designing/continuously refining novel research artifacts (von Hevner et al., 2004; Sonnenberg and vom Brocke, 2012). We conducted prescriptive research activities, making informed and well-investigated decisions in designing the research artifact of the thesis (Sonnenberg and vom Brocke, 2012). The thesis at hand answers the following main research question, which may contribute to achieving the overall objective of this research:

Research Question
How to conceptualize, design, and develop a scheduling methodology that integrates and facilitates the combined utilization of simulation, heuristics, metaheuristic, and deep reinforcement learning methods to address multi-objective scheduling problems in cloud and manufacturing environments?

To break the complexity in the design and development of the intended research artifact and answer the main research question, we follow a component-based approach (Turowski, 2003, 2001). The intended research artifact comprises four main components: a simulation component, a heuristic library component, an optimization component, and a machine learning component. Similarly, the design and development of each component are associated with sub-questions, whose answers contribute to answering the overall research question. The first sub-question explores the body of the knowledge base in search of appropriate solutions for addressing multi-objective scheduling problems in cloud and manufacturing environments.

Sub-Research Question 1
What are the key methods frequently adopted to address multi-objective scheduling problems, and how do they perform in cloud and manufacturing environments?

In addition to analyzing the performance of identified heuristic and metaheuristic methods individually, it is necessary to investigate their combination in terms of performance for solving multi-objective scheduling problems in cloud and manufacturing environments. Therefore, the second sub-question investigates the design and performance of an efficient optimization component of the research artifact. We pursue a hybrid method combining heuristics and improvement methods to leverage their advantages.

Sub-Research Question 2
How do we efficiently combine heuristic and metaheuristic methods for solving scheduling problems in cloud and manufacturing environments?

The *third hypothesis* of this research presumes that combining various heuristic or metaheuristic methods yields better performance for solving multi-objective scheduling problems during a scheduling period. The fact that most scheduling problems are NP-hard in the strong sense contributes to this assumption (Cook, 1971; Conway et al., 1967). However, for this statement to hold true, efficient integration is required. Validating the *third hypothesis* aligns with answering the previous sub-question and next sub-question. The objective is to make informed design decisions for the simulation, optimization, and heuristic library components to achieve high-quality solutions, which is the focus of the third sub-question.

Sub-Research Question 3
Will a scheduling methodology that facilitates the combined utilization of heuristic and metaheuristic methods outperform their individual use for solving multi-objective scheduling problems in cloud and manufacturing environments?

Validating the *first hypothesis* and *second hypothesis* requires investigating the adoption and integration of deep reinforcement learning methods with the respective heuristic methods to address scheduling problems. Hence, the final sub-question can be expressed in the following:

Sub-Research Question 4
How to adopt deep reinforcement learning techniques to learn from solutions to multi-objective scheduling problems?

Designed artifacts in information system and business informatics disciplines are developed following either routine design or design research practices (von Hevner et al., 2004). *Routine design* is usually adopted to apply known methods from the knowledge base while following well-established guidelines to address known business needs through a conventional IT artifact (e.g., integrating a financial solution for billing in a company) (von Hevner et al., 2004). However, *design research* must produce an innovative and unique artifact that attends to business needs with improved solutions that may be more effective or efficient in solving a given problem (von Hevner et al., 2004). In essence, the DSR artifact must address changes and meet new requirements of the application environment following scientific methods. The primary distinctions of a DSR artifact are a clear definition of the contributions to the body of the knowledge base and a viable utility in the application environment. On the one hand, translating the needs of the application environment to design useful artifacts that can be adopted in industry is fundamental. On the other hand, communicating intermediate and final results with the respective research communities is necessary to ensure consistent enrichment of the research foundations and

methodologies. The main contribution of the research artifact at hand is discussed in the next section in chronological order.

1.4 Publications of the author

The presented thesis summarizes years of extensive research and development efforts, in which over 50 scientific contributions have been published to communicate the results to the international scientific community. The thesis at hand integrates the author's most significant contributions into a single harmonized scientific artifact. As discussed earlier, we followed a component-based approach (Turowski, 2003, 2001) to conceptualize, design, and develop the research artifact. Therefore, in chronological order, we present an overview of published materials and their contribution to the individual component of the overall research artifact.

Analysis of existing methods in various application fields- Appropriate scheduling practices significantly impact the overall performance of businesses and service centers in various fields such as emergency departments (Nahhas et al., 2017b), data centers and cloud operations (Nahhas et al., 2018a), manufacturing (Nahhas et al., 2017a), or supply chain (Nahhas et al., 2023a; Daase et al., 2023). Around the beginning of the research project, we investigated conventional solution techniques often employed to deal with scheduling problems. We also studied various formulations of scheduling problems in different application fields. For instance, in Nahhas et al. (2017b), we integrated heuristic methods with discrete event simulation methods to optimize the scheduling of patients into various healthcare givers and resources such as procedures or examination rooms in an emergency department. The formulated scheduling problem was a single-stage scheduling problem, although patients with critical conditions might undergo several processes in multiple stages. The presented heuristic optimization addressed minimizing the number of patients treated before leaving the emergency department and the center's operational and staffing costs.

In the manufacturing field, we addressed a multi-stage scheduling problem considering the minimization of the makespan and the total tardiness (Nahhas et al., 2017a, 2016; Aurich et al., 2016). In this research, we developed a heuristic algorithm to deal with the problem subject to sequence-dependent setup time constraints. The research's second objective is to quantitatively investigate the performance of some metaheuristic techniques for solving multi-stage scheduling problems. Therefore, we implemented Simulated Annleaning (Kirkpatrick et al., 1983) and Tabu Search (TS) (Glover, 1989) to solve the problem given the discussed objective values. The experimental setup focuses on the performance of the solution techniques for minimizing the objective values, taking into account the required computational effort to achieve high-quality solutions. Computational results suggest a slight outperformance of the developed heuristic for minimizing the makespan while failing to achieve complete dominance compared to the metaheuristics.

In conclusion, the adopted metaheuristic techniques require significantly more com-

putational time to achieve robust results compared to the presented heuristic. Therefore, in Nahhas et al. (2017a), we investigated the adoption of population-based metaheuristics to mitigate the computational effort drawback in the previous study. The results of an initial literature analysis encouraged the adoption of Genetic Algorithms (Holland, 1975; Goldberg, 1989) as a prominent population-based metaheuristic technique. Given the same discussed problem, we evaluated the performance of the GA against the previous methods for solving four real problem instances. The computational results showed that the GA outperforms SA and TS in solving the problems and achieved better objective values. However, computational efforts remained an open challenge since only slight improvements could be made. These observations are consistent and further signified in an extended study, in which we considered additional operational constraints such as machine availability (Aurich et al., 2017). The investigation also shows that adopting population-based metaheuristic techniques guarantees an edge in performance compared to other metaheuristics since they can avoid the so-called local optima phenomena.

To address scheduling problems in a cloud environment, we investigated the performance of heuristic methods for workload management in (Nahhas et al., 2018a). In this study, we focused on collecting data to model the considered cloud environment using discrete event simulation methods. Based on the collected data and conducted interviews, a simulation model combined with heuristics was built to investigate workload patterns of 290 SAP application systems that were hosted and maintained in the considered cloud environment. The objective of the conducted study was to investigate the potential of heuristic methods to minimize the overall energy consumption of the considered environment. The experimental results demonstrated that applying heuristic methods to manage the workload of the considered cloud environment may yield up to a 70 % reduction in energy consumption. However, the study also revealed that many jobs were rescheduled during the operation, which obviously negatively impacted the performance of the customers. Optimizing multiple objective values requires employing different heuristics during a scheduling period to target every objective value independently.

State-of-the-art and literature analysis - The initial investigations into the performance of heuristic and improvement methods in the analyzed fields of applications demonstrated the importance of developing novel solutions that leverage the advantages of both techniques. The light execution time of heuristic methods, alongside the robustness and high-quality solutions of metaheuristic methods, are essential features for designing and developing advanced scheduling methods. Inheriting the advantages of these methods may allow us to mitigate their disadvantages. To confirm our preliminary observations, we investigated the state-of-the-art scheduling techniques in the fields of manufacturing and cloud in (Nahhas et al., 2018b), and (Nahhas et al., 2019b) respectively.

Nahhas et al. (2018b) investigated the influence of various technological advances in manufacturing on scheduling practices. The research focused on the introduction of Industry 4.0 initiatives and Cyber-Physical Systems (CPS) and their promised potential for the industry, which was recently further discussed by Kharitonov et al. (2024). The experimen-

tal results, which included a systematic comparison of heuristic, metaheuristic, and newly developed hybrid methods, showed that new technologies necessitate efficient scheduling solutions to achieve a competitive edge in markets. The study also covered an investigation of several real-time operational constraints, which suggested limitations of conventional heuristic and metaheuristic methods for solving complex multi-stage scheduling problems. These limitations were either in terms of solution quality or required computational effort to solve the scheduling problems. The combination of both heuristic and metaheuristic methods sustained acceptable computational effort for achieving good solutions. In this research, we emphasized the importance of multi-objective scheduling solutions to systematically close the gap between the research done in scheduling theory and adopted solutions in the industry.

Similarly, Nahhas et al. (2019b) presented a holistic view of the scheduling problems in cloud environments. In this research, we investigated how emerging IT technological advances influenced the formulation of scheduling problems and their complexity. We conducted a structured literature review to understand the impact of these technologies on scheduling problems in cloud environments and answer some research questions, such as *"How the formulation of problems changed the design of often adopted solution techniques?"*. The related literature was investigated in terms of the problem formulation, the pursued objective function, the adopted solution methods, and the architecture of the proposed solutions. The research shows that the IT industry experienced a similar industrial revolution marked by key well-established technologies. Among others, virtualization strategies, live migration algorithms, cloud computing service models and their associated advances, and major breakthroughs in ML have constituted focal points in the IT industry in the last decade.

Similar to the manufacturing field, the analysis results demonstrated that the complexity of scheduling problems significantly increased in the last decade due to new industrial requirements. Most server consolidation problems, virtual machine placement problems, dynamic virtual machine placement problems, and job scheduling in cloud environments are NP-hard combinatorial scheduling problems. This finding largely agrees with the results published in (Challita et al., 2017; Katal et al., 2023), especially if the problems are subject to multi-objective optimization measures considering performance and energy consumption. That fact excludes using exact optimization methods to solve them due to high computational efforts. Hence, most identified solutions in the literature adopted heuristic, improvement, or some hybrid methods to maximize performance or minimize operational costs. Although our investigation revealed that some machine learning-supported scheduling techniques are present in the literature, further research is necessary to investigate the development of adaptive scheduling to address the dynamic nature of these environments. In our conclusion, we emphasized the dominance of performance as the objective function and the scarcity of multi-objective solution techniques that also consider minimizing energy consumption and its associated carbon footprint.

Design and evaluation of the scheduling methodology - Based on the analysis of conventional solution methods for scheduling problems and the literature analysis, we presented initial results combining simulation, heuristic, and optimization components of the artifact in cloud environments (Nahhas et al., 2019a) and manufacturing environments (Nahhas et al., 2018b). We communicated the component-based design of the artifact and briefly discussed the integration of the simulation, heuristic, metaheuristic, and machine-learning methods for addressing scheduling problems in cloud environments. The proposal was based on the findings that we communicated in (Nahhas et al., 2017a) and (Nahhas et al., 2018a) emphasizing the complexity of multi-objective scheduling problems. In this research, we investigated whether combining different heuristic methods during a scheduling period would outperform the performance of the individual heuristic for solving scheduling problems in cloud environments to contribute to the validation of *hypothesis 2* and *hypothesis 3*. We relied on GA to implement an overreaching control strategy to switch between heuristics during the scheduling period. The collected results demonstrated substantial improvement in minimizing energy consumption and the total number of rescheduled jobs during the scheduling period. The experimental analysis showed that the proposed methodology adapted to workload changes and switched between heuristics during the scheduling period to leverage their suitability under various workload conditions.

To investigate the performance of the scheduling methodology on large-scale problems, we conducted an extensive evaluation for solving single-stage and multi-stage scheduling problems in cloud (Nahhas et al., 2021a) and manufacturing (Nahhas et al., 2021b) environments. In (Nahhas et al., 2021a), we also adopted a GA to control different heuristics during a scheduling period. To establish a comparison, we designed the experiments using real-world cloud workloads based on the known PlanetLab benchmark. The experiments are subject to the same conditions and problem formulation presented in (Beloglazov et al., 2012a) and (Moges and Abebe, 2019). The problems were solved with the objective of minimizing energy consumption and violations in Service Level Agreements (SLA). The collected results demonstrated that the scheduling methodology contributes to minimizing energy consumption by at least 30 % and reaching up to 47 % compared to several heuristics proposed in (Beloglazov et al., 2012a; Moges and Abebe, 2019). However, the results also showed inferior performance in terms of minimizing the SLA violation, amounting to a 0.04 % increase compared to the same heuristic methods. The considered objective measures are obviously of a conflicting nature. It is for a decision maker to evaluate whether a 47 % reduction in energy consumption justifies a 0.04 % increase in SLA violations.

Similarly, we carried out an extended evaluation of the scheduling methodology for solving multi-stage scheduling problems in the field of manufacturing in (Nahhas et al., 2021b). Multi-stage scheduling problems are inherently more complex than single-stage ones since scheduling decisions in the first stage significantly impact the system's overall performance. We evaluated the presented methodology for solving four-stage HFS scheduling problems considering four objective values: the minimization of the makespan, the total tardiness, the number of penalties, and the number of major setup times. The evaluation is conducted to solve thirty problem instances provided by our partner. The

problem instances are extracted based on the production backlog of a manufacturing system in printed circuit board production. The presented scheduling method achieved complete dominance compared to known heuristics in the literature for solving 77 % of the problems and found the best solutions to minimize all objective values. In 23 % of the problem instances, the scheduling method found solutions that partially dominate GA for minimizing at least three objective values. Regarding computational effort, it maintained a robust performance, with up to 10 times quicker searching for reported solutions than GA. However, parallelization could further reduce the required computational effort to support real-time decision-making processes.

The author of this thesis supervised a master thesis to investigate further the potential of the approach using the NASA and KTH cloud workload traces (Dror G. Feitelson et al., 2014). These traces also included thousands of jobs that must be scheduled in a single-stage cloud environment. Based on the same experimental setup, the obtained computational results showed that the presented method dominates baseline heuristic methods for solving the problems in terms of minimizing the makespan, the average flow time, and the average waiting time. A detailed analysis of the computational results indicated that the potential improvement increases as the number of jobs that must be scheduled increases. It implies that the method's outperformance of heuristics would increase with the increase in problem complexity. The results were presented in (Remesh et al., 2022) and (Remesh et al., 2023). The first study focused on investigating the technique's performance given various combinations of objective values such as the makespan, the average waiting time, the throughput, and the average flow time (Remesh et al., 2022). The second study concluded the final results (Remesh et al., 2023).

In previous evaluations, we formulated multi-objective scheduling problems and solved them as mono-objective using a weighted approach, which limits the search process. Hence, to address this limitation, we relied on a pure multi-objective control strategy and replaced the GA with the Non-Dominated Sorting Genetic Algorithm three (NSGA III) in Nahhas et al. (2022a). To fully harness the potential for scalability in modern hardware, we developed a parallelization layer, which distributes the evaluation of solution individuals on available physical resources. The main components of the artifact were already developed using open-source technologies for simulation and parallelization. We collaborated with colleagues to investigate the use of open-source discrete event simulation core as an alternative for commercial simulation packages in (Lang et al., 2021a). Finally, we validated the intended improvements of the methodology for solving multi-stage HFS problems from related works. The computational experiments showed that the pure multi-objective optimization outperformed all previously presented best solutions in (Nahhas et al., 2022a) and (Lang et al., 2020) for minimizing four considered objective values. Given the discussed results, it is evident that pure multi-objective optimization explores the solution space significantly better than the weighted-sum approach.

Meanwhile, research and development efforts were committed to designing and developing the machine-learning component of the artifact. We presented the evaluation of the DRL scheduling model and its integration into the artifact in (Nahhas et al., 2022b). The

first objective of this research was to investigate the first research hypothesis and propose a prototypical implementation of the machine learning component of the research artifact. The second objective was to evaluate the design and the performance of the machine learning component of the presented artifact. We adopted two prominent DRL techniques, Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor-Critic (A3C), for solving multi-stage scheduling problems considering multi-objective values. The presented prototypical implementation relied on the same simulation engine used in (Nahhas et al., 2021b) to the machine learning component with the rest of the artifact components. The results of our computational analysis demonstrated that the presented method successfully approximates appropriate scheduling policies. To establish a comparison, we replicated a two-stage scheduling problem, investigated in (Lang et al., 2021b), and evaluated the performance of the DRL methods to solve them. The computational results showed that the A3C is more stable than PPO, especially when agents are exposed to unknown problems. Compared to NEAT presented in (Lang et al., 2021b), the A3C delivers superior solutions in terms of minimizing all objective values for solving three problem instances while sharing comparable solution quality with NEAT for solving the fourth one. *The presented research was nominated and awarded the Best Paper award by the International Scientific Committee of a top-ranked scientific conference in the analytic and decision science track.*

In conclusion, the empirical analysis suggests that DRL methods can learn from solutions to multi-stage scheduling problems. However, we encountered generalization issues when DRL agents were exposed to unknown, significantly different problems that led to a slight degradation in the quality of solutions. DRL agents steadily recover after further training and adjust their scheduling policy accordingly. Our observation of the generalization matter largely agrees with the results published by Google Brain and DeepMind in their paper on DRL methods (Zhang et al., 2018) and later extended in (Zhang et al., 2021).

Therefore, in Nahhas et al. (2024a), we presented an investigation of the generalization issue and communicated a service-oriented architecture of the scheduling methodology. Based on our previous work in Nahhas et al. (2022b), we demonstrated the use of Imitation Learning (IL) principles by combining the utilization of multi-objective optimization methods and DRL methods to propose a (DRL-based IL). The notion of imitation learning strives to utilize expert knowledge to achieve behavioral cloning. In other words, a DRL agent imitates desired behavior, cloned by an expert, such as an optimization component, for solving a scheduling problem. Therefore, we adopted the Monotonic Advantage Re-Weighted Imitation Learning (MARWIL) method (Wang et al., 2018) in combination with the NSGA III (Nahhas et al., 2022a) as an expert policy to address multi-objective scheduling problems. The computational results of the study confirmed that the DRL-based IL achieves superior results compared to the pure DRL technique. With up to ten percent higher mean reward, the DRL-based IL partially mitigates generalization issues when agents are exposed to solving completely unknown scheduling problems.

In Nahhas et al. (2024b), we investigated the problem from another perspective and

adopted an image-based observation space for training A3C and PPO DRL methods. The methods were used to solve scheduling problems in a two-stage supply chain retailing environment subject to multiple objective values. In this research, we encoded the collected observations from the simulation component in images and passed them to the DRL agents in addition to rewards during training. The computational results demonstrated that the agents learn to solve schedule product families to retail stores, taking into account the minimization of the average time of selling and the makespan.

Other works that contributed to design and evaluation decisions of the artifact - The author collaborated with many researchers on scheduling problems and related fields. We will briefly discuss some published works in chronological order to which the author contributed.

Objective values for solving scheduling problems in manufacturing, such as the maximum completion time, the total tardiness, or the machine utilization, are straightforward optimization measures that have been well-studied in the literature in the last decades (Baker and Trietsch, 2009; Graham et al., 1979; Conway et al., 1967). However, they are insufficiently investigated in cloud environments (Pires and Barán, 2015). For decades, the performance of an IT system has been overwhelmingly the sole concern of IT service providers until recent environmental crises and current energy entanglements (Koomey, 2011; Katal et al., 2023).

Therefore, we systematically analyzed different functional and non-functional requirements models of performance in IT systems (Alwadi et al., 2018). The objective was to construct relevant numerical performance models which can be used for optimization. We concluded the analysis with a proposal for a new performance requirements model of the IT system. However, the interdependencies between the identified performance requirements remained an open issue, which we addressed in (Alwadi et al., 2019). In this research, we highlighted the importance of certain requirements, such as scalability, efficiency, and resource utilization. *These requirements were significant in designing the artifact of this thesis, as we will discuss in Chapter 3.* We evaluated the presented performance requirements model, which included functional and non-functional requirements, in a survey distributed to IT experts and practitioners.

In cloud environments, shortly after introducing a new carbon emissions taxation law, we investigated the potential of scheduling methods for load consolidation to minimize costs and carbon emissions. We relied on real-world workloads of 20 data centers hosting standard enterprise systems (Bosse et al., 2020). To solve the single-stage scheduling problems, we investigated the use of heuristic, metaheuristic, and hybrid methods considering five virtual machine types, five tax levels, and two power mixes (fossil and renewable). Computational results demonstrated that significant optimization potential can be achieved, especially if a high-emissive power source is used for operating a cloud environment.

In manufacturing, Rolf et al. (2020a) adopted the GA algorithm to select different PDRs for solving two-stage scheduling problems with family setup time constraints based

on Nahhas et al. (2017a) and Nahhas et al. (2018b). In this implementation, the GA decides which heuristic should be used for scheduling every job. The presented technique is evaluated for solving four problem instances to minimize the makespan and total tardiness. The computational results showed that the presented scheduling method entirely dominates the individual heuristic techniques for solving the problems. To increase the quality of achieved solutions, Rolf et al. (2020b) applied the concept of decision points between heuristics following the same design presented by Nahhas et al. (2018b) and Nahhas et al. (2019a). The experiments showed that the presented approach achieves high-quality solutions, superior to the metaheuristic methods presented in (Aurich et al., 2017).

To achieve automatic simulation model construction in the presented artifact of the thesis, we had to rely on an open-source discrete event simulation engine. Lang et al. (2021a) present a systematic comparison between several open-source simulation engines and other counterpart commercial simulation packages. The results demonstrated that relying on an open-source simulation engine grants engineers higher flexibility in combining developed models with machine learning and/or other optimization technologies.

In (Müller et al., 2022), server consolidation problems were investigated to minimize energy consumption while addressing performance concerns. This work relied on machine learning techniques as a supporting component to an optimization component for approximating the performance of new system placement instead of using a simulation model. The reason behind such practice is to minimize the modeling effort required to build simulation models for complex IT system landscapes. The use of machine learning techniques for addressing predictive business concerns, such as detecting anomalies or workload spikes that cause performance degradation, was also investigated. Similarly, we analyzed different conventional machine-learning methods for detecting anomalous behavior in manufacturing systems in (Kharitonov et al., 2022). Production logs were used to detect machine breakdowns, which might cause violations in delivery dates. The empirical results compared the performance of ten conventional machine learning methods. The computational results concluded with recommendations on using certain conventional methods that performed best to detect anomalies in manufacturing environments.

A detailed description of the designed research artifact and its underlying functionalities will be discussed in Chapter 3. The following section will provide an overview of the structure of this research.

1.5 Thesis structure

This thesis is structured into five distinct chapters and follows the design science research framework. The current chapter started with a motivation for the thesis and articulation of the research gap. Based on the research gap, the research design based on the DSR framework was thoroughly discussed to designate the objectives of the thesis from research and development perspectives.

The second chapter presents the reader with theoretical foundations and literature analysis of the work. It is structured in six sections and starts with an overview of schedul-

ing foundations and preliminaries. Then, we present and discuss the methods that we adopted and integrated to design the thesis's artifact in chronological order: scheduling, simulation, heuristic, metaheuristic, and DRL methods. We start with an introduction to simulation methods. Then, we thoroughly discuss the conventional solution methods used to address scheduling problems with a primary focus on heuristic and metaheuristic methods. Afterward, we present the required foundation and preliminaries of DRL in general. Finally, the results of the literature analysis are presented and thoroughly discussed before concluding the chapter with a summary that refers back to the research gap.

The third chapter thoroughly discusses the overall design of the presented artifact. It incorporates six sections and follows the same logical flow in the the second chapter. The objective of this chapter is to elaborate on the design and integration of the artifact's components. The motivation and research gap of the thesis, supported by the findings of the theoretical foundations, are translated into design requirements. The derived requirements are grouped into functionality layers, which conclude the requirements stage and introduce the foundation of the design chapter. Based on the functionality layers, the design of the thesis artifact is presented using component and modular design practices. The reader is first presented with the design of the modeling and simulation components, followed by the design of heuristic library components, and finally, the design of the optimization and machine learning components. Throughout the design chapter, we discuss the intermediate results to summarize evaluation activities that are conducted during the design of the artifact.

The fourth chapter presents the conducted results and the analysis of the conducted evaluation activities. This chapter is structured based on the evaluation activities suggested by Sonnenberg and vom Brocke (2012) and follows the adopted research framework presented by von Hevner et al. (2004). The chapter encompasses five sections and starts with a summary of the first evaluation activity. The first activity concludes the state-of-the-art analysis and supports the discussed research gap. The second activity elaborates on evaluating the adopted design tools and their suitability to produce the blueprints of the artifact's components and their integration. Afterward, the implementation of the artifact is presented as a proof of concepts to evaluate the proposed methodology following the DRS framework (von Hevner et al., 2004). Section 4.3 discusses the implementation and deployment of the artifact, which integrates the utilization of simulation, heuristic, metaheuristic, and DRL methods to address multi-objective scheduling problems. It provides an overview of the most significant adopted technologies in the presented methodology to achieve parallelization, scalability, and efficient multi-architecture execution.

After summarizing the implementation and deployment, the artifact is instantiated to solve single-stage and multi-stage scheduling problems in cloud and manufacturing environments. Throughout the third and partially fourth evaluation activities, the artifact's applicability and performance are measured in solving real multi-objective scheduling problems using real problem instances. We concluded each evaluation phase with a summary that highlights the main findings of the collected results. Finally, *the fifth chapter* recaps the presented thesis and highlights the main findings and scientific contributions.

2 Theoretical foundation and literature analysis

This chapter is structured into six sections that present the reader with an overview of scheduling problems and an analysis of the literature. The first section discusses the necessary preliminaries of scheduling problems. The second section is dedicated to describing modeling and simulation methods that are necessary for designing the intended artifact. The third section presents a holistic view of solution methodologies that are often adopted to solve scheduling problems. It covers heuristic and improvement techniques and their adoption in addressing scheduling concerns. The fourth section highlights the required preliminaries of deep reinforcement learning methods for designing the intended methodology. To summarize, the reader is presented with an overview of scheduling problems, simulation, heuristics, improvement, and machine learning methods. Then, the fifth section concludes the theoretical foundation chapter with a structured analysis of the literature. Finally, the last section summarizes the findings of this chapter.

2.1 Scheduling

A scheduling problem can be expressed using a three-field tuple $\langle \alpha \mid \beta \mid \gamma \rangle$ as suggested by Graham et al. (1979) and further used in (Pinedo, 2012, p. 14). The first field, α denotes the machine environment and configuration. The second field, β, represents the characteristics of jobs and the operational constraints that must be considered. The third field, γ, expresses the perused objective functions for solving a given scheduling problem. In addition to the three-field tuple representation, one must present and discuss the required notations for the data that describes jobs, such as processing times on every stage, deadline, or required capacities (Graham et al., 1979; Pinedo, 2012, p. 14). This section provides the required preliminaries for formulating single- and multi-stage scheduling problems. For further reading references on job shop scheduling, one might refer to the works presented by (Pinedo, 2012), (Baker and Trietsch, 2009), and (Graham et al., 1979).

2.1.1 Variations of scheduling problems

A scheduling problem depends strongly on the structural shape of a considered cloud or manufacturing environment and is denoted in the $\alpha = \alpha_1\alpha_2$ field. The α_1 describes the type of physical machines that are available for processing different jobs. The α_2 dictates the number of available physical machines for processing jobs. Depending on the opera-

tional nature of a considered environment, jobs either require certain resources on a single machine at a single stage to be completed or might undergo multiple stages to be processed using several machines before completion. The former type of scheduling problem is very common in cloud environments, in which a set of virtual machines must be scheduled on a set of available physical machines given some objective function. Multi-stage scheduling problems are inherently more complex. They are more common in manufacturing environments and workflow scheduling in cloud environments. Both types of scheduling problems often encompass an underlying allocation and a sequencing sub-problem (Baker and Trietsch, 2009, p. 221). The allocation and sequencing dimensions of a scheduling problem are usually formed by the description of the machines within a considered system. In scheduling theory, three types of machine descriptions are often used: identical parallel machines, uniform parallel machines, and unrelated parallel machines.

For any scheduling problem, we may assume, given a scheduling period, that the number of jobs and the number of available machines are known and finite till the next point in time. The number of jobs in scheduling theory is usually denoted by n and the number of machines by m. To define the rest of the mathematical sets, we bound the size of a set using cardinality. We will also use the subscript i to refer to an arbitrary operation or machine and the subscript j to refer to an arbitrary job. Given a set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ that must be scheduled. We also have a set of machines $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$ that are available to process jobs. A job $J_j \in J$ consists of a set of operations $O_j = \{O_{o,j}, \ldots, O_{|O|,j}\} : \forall\ o \in \{1, \ldots, |O|\}$. Each operation $O_{o,j}$ of a job J_j is associated with a processing time and is denoted by $p_{i,j} \in \mathbb{R}^+$. Every operation must be completed on a machine $M_i \in M$. Based on these introductory notations, we will discuss the most important machine configurations in scheduling practice. We generally rely on the works presented by Pinedo (2012), Baker and Trietsch (2009), and Graham et al. (1979).

Single machine scheduling problem $\alpha_1 = 1$:

The single-machine scheduling problem is the simplest variation of scheduling problems in the literature, and it has also been studied the most. As the name suggests, a set of jobs $J_j \in J$ must be scheduled on a single machine $M = m = 1$ to be processed subject to achieving some objective(s). Solving a single-machine scheduling problem degenerates into solving a sequencing problem where the order of processing jobs is optimized to maximize or minimize some objective function.

Identical parallel machines scheduling problem $\alpha_1\alpha_2 = P_m$:

In such environments, a set of identical machines is available to process jobs in parallel. Every job $J_j \in J$ consists of a single operation $O_{i,j}$ that must be processed on a machine $M_i \in M$. Since the processing time of a job is identical on all machines, it is denoted by p_j omitting the i subscript that points to a machine. Identical parallel machine scheduling problems are very common in cloud environments.

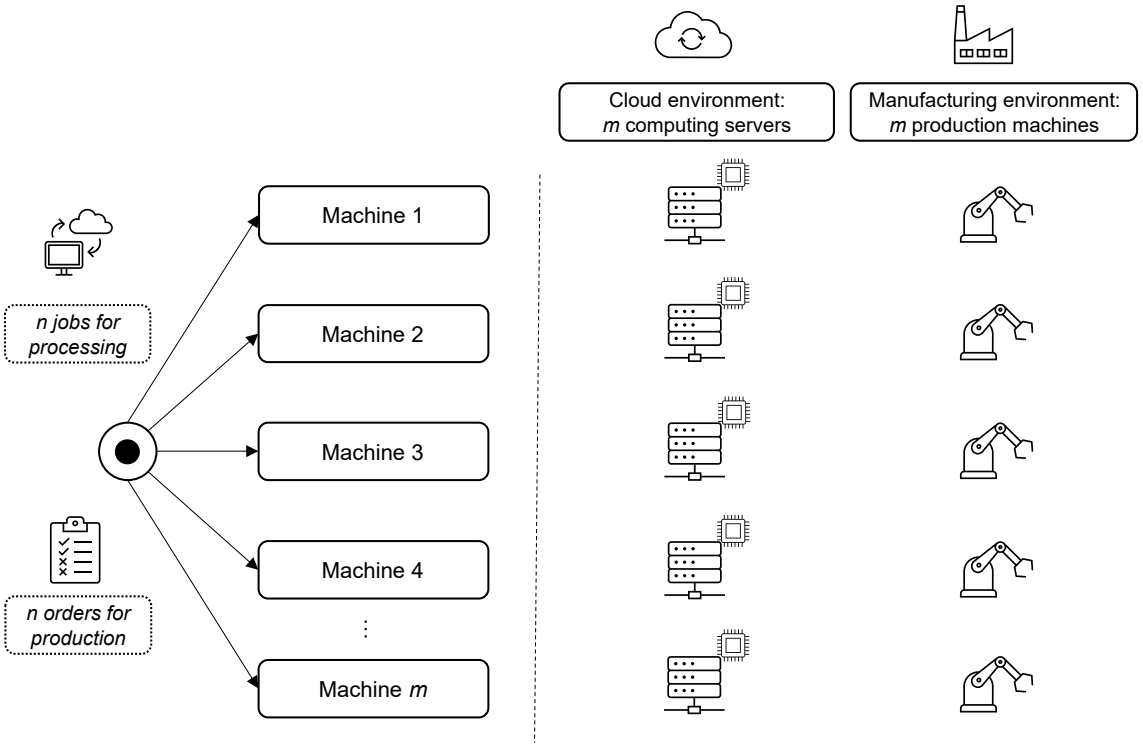


Figure 2.1: Parallel machines scheduling in a cloud or a manufacturing environment.

For instance, a set of virtual machines or jobs must be scheduled on a set of identical physical machines.Figure 2.1 demonstrates a classical parallel machine scheduling problem in a cloud or a manufacturing environment. Solving P_m scheduling problems underlines dealing with allocating jobs to machines and optimizing the sequence in which jobs are processed on every machine (Baker and Trietsch, 2009, p. 221). During the design and development phase of the research artifact, we addressed several P_m scheduling problems (Nahhas et al., 2019a; Nahhas et al., 2018a).

Uniform parallel machines scheduling problem $\alpha_1\alpha_2 = Q_m$:

It is a generalization of the identical parallel machines problem. Every job $J_j \in J$ also consists of a single operation, which must be carried out on one of the parallel machines $M_i \in M$. However, these machines' processing speeds may be different, resulting in a different processing time $p_{i,j}$, which refers to the machine M_i that processed the job. In this thesis, we evaluated several components of the research artifact for solving different variations of Q_m scheduling problems in cloud environments in (Remesh et al., 2023, 2022; Nahhas et al., 2021a, 2019a).

Flow shop scheduling problem $\alpha_1\alpha_2 = Fm$:

There is a set of machines $M_i \in \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$ that are available in series. Given a set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$, every job J_j must be processed following the exact technological order to complete all operations.

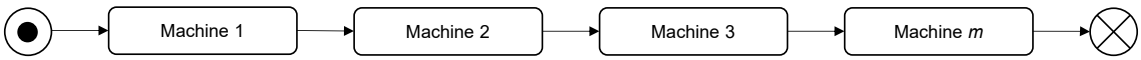


Figure 2.2: A classical flow shop scheduling environment.

For instance, every job must be processed on the first machine, followed by the second and third until all operations are completed as depicted in Figure 2.2. This type of problem is also common in cloud environments, where a workflow must be scheduled on computing resources given a strict technological order of operations.

Hybrid Flow shop scheduling problem $\alpha_1\alpha_2 = HFSm$:

The Hybrid Flow Shop Scheduling problem (HFS) is a generalization of the parallel machines and flow shop scheduling problems. In these environments, there is a set of processing stages $S = \{S_s, \ldots, S_{|S|}\} : \forall\ s \in \{1, \ldots, |S|\}$. At every stage S_s, a set of parallel machines $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$ are available to process all types of jobs. In such environments, the operations $O_j = \{O_{o,j}, \ldots, O_{|O|,j}\} : \forall\ o \in \{1, \ldots, |O|\}$ of a job J_j are completed sequentially at the processing stages using one of the available parallel machines. All jobs must follow the same technological order, meaning all jobs are first processed in the first stage, then the second, and finishing, for example, with the third using one of the available machines. Figure 2.3 represents a classical HFS scheduling environment with multiple stages. HFS are fairly more complex problems than the previously discussed types. They are more common in manufacturing than in cloud environments. In the Literature, they are interchangeably referred to as "flexible flow shop" in industrial engineering literature or "multi-processor flow shop" in computer science literature (Pinedo, 2012, p. 15).

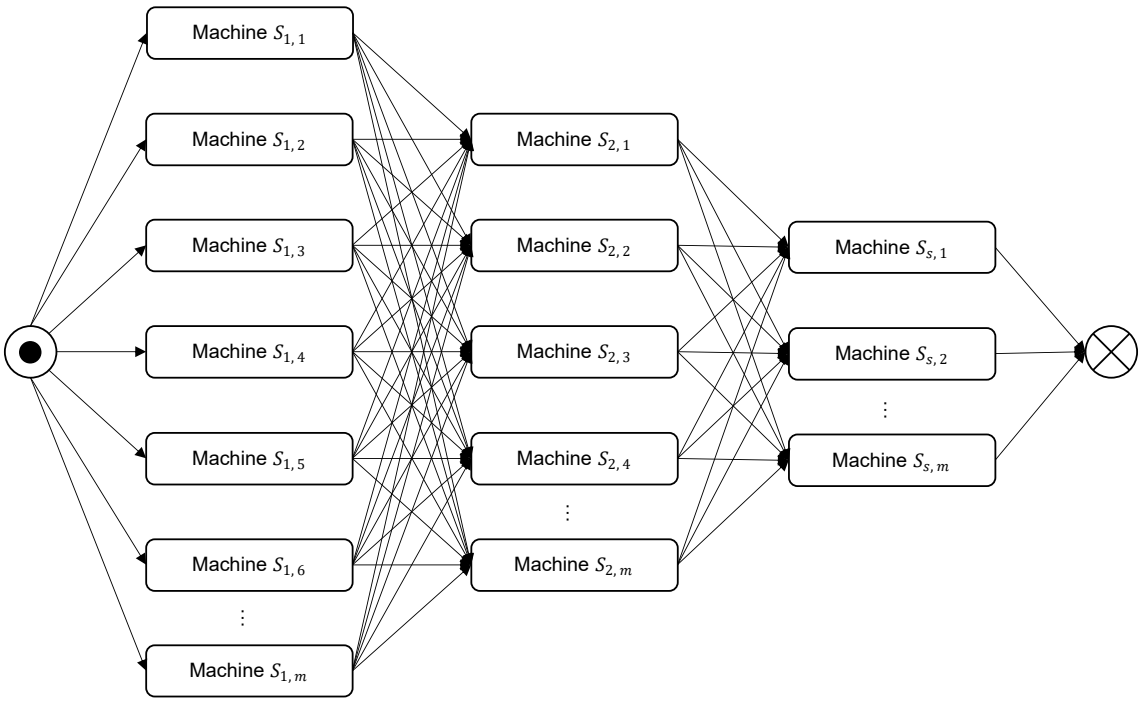


Figure 2.3: Hybrid flow shop scheduling environment.

In the thesis at hand, we investigated several real-world HFS_m scheduling problems in manufacturing. We extensively evaluated the performance of the artifact's machine learning, optimization, and simulation components in Nahhas et al. (2022b), Nahhas et al. (2021b), Nahhas et al. (2017a) respectively.

Job shop scheduling problem $\alpha_1\alpha_2 = Jm$:

In job shop environments, there are also $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$ that are not subject to any processing stages or technological orders. Every job $J_j \in J$ has its own set of operations and associated processing routes on the machines. Additionally, all jobs must be processed at one of the machines at least once and may also be processed on the same machine several times. In this case, the recirculation constraint must be taken into account. In this work, we do not address job shop scheduling problems. We briefly describe them since some works in the literature, which we will discuss in the thesis, deal with job shop problems.

2.1.2 Operational constraints

Release date $\beta = r_j$:

The release date of a job $J_j \in J$ is denoted by r_j and indicates the time of the job being released for scheduling (Sousa and Moreira, 2007) and (Pinedo, 2012, p. 15). For instance, in cloud environments, periodical jobs such as updates, maintenance, and security are already planned but not released for scheduling up to a specific point in time. In the manufacturing context, considering release date constraint is usually part of the scheduling model when the objective function involves the minimization of inventory costs (Sousa and Moreira, 2007; Ebben et al., 2005). For instance, despite available resources for processing a job, starting early with processing an order leads to an increase in invested capital for raw materials and higher inventory costs (Baker and Trietsch, 2009, p. 16-17). After discussing common objective values in solving scheduling problems, we will elaborate more on the release date constraint. In the majority of studied problems, we pursued the minimization of delivery date violations, which is indirectly affected by this constraint (Nahhas et al., 2022a, 2017a). However, we did not consider it to be a hard constraint in our problem formulation.

Preemptions $\beta = prmp$:

Including this constraint in the formulation of a scheduling model implies allowing jobs that are already in processing to be interrupted. The interrupted operation $O_{i,j}$ of a $J_j \in J$ on the machine $M_i \in M$ may be resumed later during the scheduling period. That means the invested processing time is not lost, and the processing might be completed on the same machine or another available parallel machine. For instance, pausing batch workload processing in cloud environments is usual when a higher-priority job is released for scheduling. In manufacturing, however, it is not typical to allow interruption of processing

a job since it negatively impacts system efficiency due to machine setup and preparation time (Pinedo, 2012, p. 16).

Precedence $\beta = prec$:

The precedence constraints in scheduling restrict mainly the sequencing part of a scheduling problem by imposing dependencies between jobs. For instance, a job $J_j \in J$ may not be scheduled unless J_{j-1} is completed. In cloud environments, it is usually a hard constraint in some workflow scheduling problems because some types of workflows are decomposed into smaller interdependent jobs that must be processed on some computing resources (Jayanetti et al., 2022, p. 15). In manufacturing environments, it is in working station manufacturing plants.

Sequence dependent setup times $\beta = s_{j,k}$:

The sequence dependency constraint is present when two jobs $J_j \in J$ and $J_k \in J$ require certain configurations on a processing machine $M_i \in M$. This configuration time is required to prepare the machine after finishing the job J_j to start processing the job J_k. It is denoted by $s_{j,k}$, which is dependent on the types of jobs and their exact position in a sequence on a machine. This constraint substantially complicates a scheduling problem since the overall system efficiency heavily depends on minimizing the overall setup times during the scheduling period (Graham et al., 1979; Lenstra et al., 1977). The majority of scheduling problems that include the setup times constraint are NP-Hard problems (Gupta and Kyparisis, 1987; Graham et al., 1979; Lenstra et al., 1977). This constraint is significantly more relevant in manufacturing than in cloud environments.

Family dependent setup times $\beta = fmls,\ f_{g,h}$:

In some scheduling environments, we may have a set of families $f = \{f_g, \ldots, f_{|f|}\} : \forall\ g \in \{1, \ldots, |f|\}$. Similar to the sequence-dependent setup times constraint, an $f_{g,h}$ amount of time is required to reconfigure the machine after finishing a job $J_j \in f_g$ to start processing a job that belongs to another family $J_k \in f_h$. Early contributions on scheduling problems with family setup times date back to the early nineties by (Wittrock, 1990; Kut C. So, 1990). Both studies addressed scheduling problems with major and minor setup times. This constraint is also more evident in manufacturing environments than in cloud ones. As the name suggests, jobs of certain types sharing, for instance, raw materials, are grouped into families. Switching a machine after finishing a job to start processing the next one within the same family requires minimal reconfiguration time. However, significantly more preparation time is required when a machine must be prepared to process a job from another family than the current one. Since the majority of assembly manufacturing environments are characterized by family-dependent setup time, we investigated this constraint in several variations of multi-stage scheduling problems in Nahhas et al. (2022a) and Nahhas et al. (2021a).

Machine breakdowns $\beta = brkdwn$:

As the name suggests, indicating the *brkdwn* constraint in the β field implies that the availability of a machine is subject to interruptions during a scheduling period. This constraint on scheduling problems is not often considered in the cloud and manufacturing literature. It is usually treated in an independent stream of research that deals with predictive or preventive maintenance problems, as presented by Wongchai et al. (2022) or Allaoui and Artiba (2004). In the context of this thesis, we studied the impact of machine breakdowns on the performance and robustness of the optimization component in dealing with multi-stage scheduling problems in Nahhas et al. (2018b).

Machine eligibility restrictions $\beta = M_j$:

The M_j constraint is declared in the β field to model the incapability of some machines to process some types of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ during the scheduling period. Here, the $M_j \subset M$ denotes a set of machines in a P_m or Q_m environment, which is capable of processing a job J_j. For instance, we investigated scheduling problems, taking into account the constraint of family setup times. When a job $J_j \in f_g$ is being processed on some machine $M_i \in M$, the remaining machines on the same processing stage are not eligible to process any other job from the same family as, for instance, $J_k \in f_g$. This constraint can contribute to a higher complexity of a considered scheduling problem (Pinedo, 2012, p. 17).

Machine capacity constraint $\beta = M^C$:

The M^C indicates that every machine $M_i^C \in M^C$ is associated with a set of resource capacities C. In such a scheduling environment, every job $J_j^R \in J^R$ is also associated with a set of resource requirements R. A job J_j^R can be processed on a machine M_i^C if the set of resources capacities in a machine C can satisfy the set of resources requirements R subject to the conditions in Υ as presented in Equation 2.1. In Equation 2.1, $|C|$ and $|R|$ denote the number of capacity dimensions in a machine and the number of requirement dimensions of a job, respectively, based on the cardinalities of their sets. This constraint is fundamental for the majority of scheduling problems in cloud environments. For instance, a virtual machine is usually associated with computational requirements such as main memory capacity, number of CPU cores, and storage capacity (Lopez-Pires and Baran, 2015; Pires and Barán, 2015). Depending on the operation of a cloud environment, a virtual machine may be allocated to a physical machine only if this physical machine has enough capacity for each of the requirements. Many similar problems are reduced to bin-packing problems (Akhter and Othman, 2016, p. 1171). In scheduling theory, other constraints such as batch processing: $\beta = batch(b)$, Blocking: $\beta = block$, No-wait: $\beta = nwt$, Permutation: $\beta = prmu$, and Recirculation: $\beta = rcrc$ might be relevant for developing solutions for scheduling problems. For further reading, one can refer to the works presented by Brucker (2007, p. 4-5) and Pinedo (2012, p. 16-17).

$$
\begin{aligned}
C &= \{C_c, \ldots, C_{|C|}\} \subset \mathbb{R}^+ : \forall\, c \in \{1, \ldots, |C|\} \\
R &= \{R_r, \ldots, R_{|R|}\} \subset \mathbb{R}^+ : \forall\, r \in \{1, \ldots, |R|\} \\
\textit{Subject to} &: \Upsilon = \{(R_r, C_c) \mid R_r \in R \textit{ and } C_c \in C \textit{ and } R_r \leq C_c\} \subseteq R \times C
\end{aligned} \tag{2.1}
$$

2.1.3 Objective values

After formalizing the structural characteristic of a considered system and declaring the considered operational constraints, the objective function is expressed in the γ field based on the triple notation $\langle \alpha \mid \beta \mid \boldsymbol{\gamma} \rangle$ presented by Graham et al. (1979, p. 288). Possible schedules are explored and evaluated based on a defined objective function that expresses some objective values (Pinedo, 2012, p. 18; Baker and Trietsch, 2009, p. 12). Before discussing the most prominent objective measures in the field of scheduling, we will present some preliminaries that describe jobs and ultimately define the objective values of the scheduling problem.

Preliminary notations

- **Waiting time W_j:** The waiting time for a job $J_j \in J$ is defined by the amount of time while it is awaiting dispatching by some machine $M_i \in M$ in a single-stage scheduling environment, for instance, P_m. In a multi-stage environment such as a HFS_m, it is the accumulated waiting time of all operation $O_j = \{O_{o,j}, \ldots, O_{|O|,j}\} : \forall\, o \in \{1, \ldots, |O|\}$ to complete a job J_j as presented in Equation 2.2.

$$
W_j = \sum_{i=1}^{n} W_{i,j} : \forall i = 1, \ldots, n \tag{2.2}
$$

- **Completion time C_j:** The completion time of a job J_j can be computed by summing up the waiting time and processing time of all operations O_j in addition to its release time as presented in Equation 2.3 based on (Conway et al., 1967, p. 11). It is, in essence, the point in time the job left the system after all its associated operations are completed (Pinedo, 2012, p. 18).

$$
\begin{aligned}
C_j &= r_j + W_{1,j} + p_{1,j} + W_{2,j} + p_{2,j} + \cdots + W_{i,j} + p_{i,j} \\
&= r_j + \sum_{i=1}^{n} p_{i,j} + \sum_{i=1}^{n} W_{i,j} : \forall i = 1, \ldots, n
\end{aligned} \tag{2.3}
$$

- **Flow time F_j:** It is the amount of time a job $J_j \in J$ spends inside a scheduling environment from the moment it is released for scheduling as presented in Equation 2.4.

$$\text{Flow Time } F_j = C_j - r_j \tag{2.4}$$

It is computed by subtracting the release date r_j of a job J_j from its completion time C_j.

- **Lateness L_j:** Let d_j be a time unit during a scheduling period that dictates the due date of a job $J_j \in J$. The lateness of a job J_j is computed by subtracting the due date from the completion time of a job as presented in Equation 2.5 (Baker and Trietsch, 2009, p. 12; Pinedo, 2012, p. 18). If the value is negative, it means that the job was processed earlier than its due date. The result is positive if a job is completed after its due date. Usually, we want to pursue a value of lateness that is as near as possible to the due date to avoid penalties and may contribute to minimizing inventory costs (Kianpour et al., 2021, p. 361).

$$\text{Lateness } L_j = C_j - d_j \tag{2.5}$$

- **Tardiness T_j:** The tardiness of a job $J_j \in J$ is a computed based on it's due date d_j and completion time C_j as presented in Equation 2.6 (Pinedo, 2012, p. 18). In essence, a job is tardy if it is completed after the due date.

$$\text{Tardiness } T_j = \begin{cases} (C_j - d_j) \text{ if } (C_j - d_j) > 0 \\ 0 \text{ Otherweise} \end{cases} \tag{2.6}$$

- **Unit penalty U_j:** Based on the definition of the lateness and tardiness of a job $J_j \in J$, a unit penalty is usually computed for every violation in the delivery date of jobs during a scheduling period as presented in Equation 2.7.

$$\text{Unit penalty } U_j = \begin{cases} 1 \text{ if } C_j > d_j \\ 0 \text{ Otherweise} \end{cases} \tag{2.7}$$

Generally, violations of delivery dates in both cloud and manufacturing environments are associated with financial penalties, deterioration of customer satisfaction, and even loss of reputation. Figure 2.4 present a summary of the previously discussed terms. The left-upper side of the figure depicts the negative or positive L_j of a job $J_j \in J$ depending on its completion time and due date.

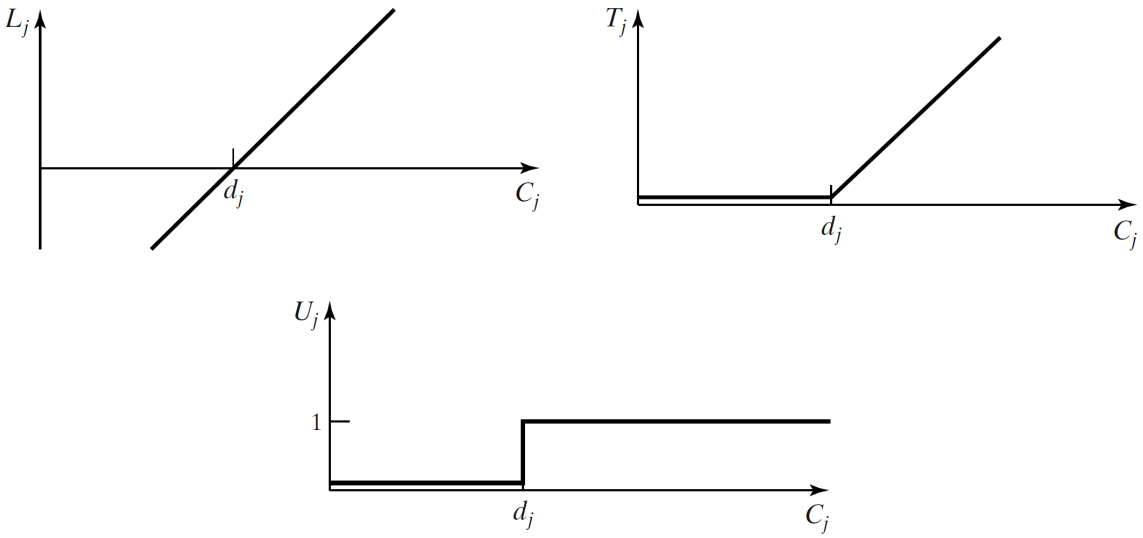


Figure 2.4: Lateness, tardiness, and penalty functions (Pinedo, 2012, p. 18).

The right-upper side of the figure explains the tardiness T_j of the job, which is always positive if the job is completed past its due date. Finally, the lower part of the figure summarizes a simple penalty function, which is usually extended and multiplied by actual financial costs associated with violations of delivery dates or some SLAs.

The makespan $\gamma = C_{max}$:

The makespan is the maximum completion time among the set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ during a scheduling period. It can be identified based on the completion times of all jobs relying on Equation 2.3 and is presented in Equation 2.8. It is basically the completion time of the last job during a scheduling period (Pinedo, 2012, p. 18; Brucker, 2007, p. 6). Based on the presented preliminaries, the first and most pursued objective function in the scheduling literature is the makespan (Neufeld et al., 2016, p. 70).

The popularity of the makespan is associated predominantly with the parallel machines problem variations. It is pretty intuitive to pursue the minimization of the makespan in a P_m scheduling problem, especially in cloud environments, since it would indirectly contribute to a workload distribution and balancing between the machines (Baker and Trietsch, 2009, p. 221). The workload balance between machines is sometimes adopted to address scheduling problems, especially in cloud environments. In this thesis, we will refer to the Workload Balance between machines in a scheduling environment by M_{LB}.

$$\text{Minimize } C_{\max} = \max C_j : \forall\ j \in \{1, \ldots, n\} \tag{2.8}$$

The mean flow time $\gamma = \overline{F}$:

Based on the preliminary notations and the flow time of a job $J_j \in J$ in Equation 2.4, the mean flow time of a scheduling problem is computed by dividing the sum of flow

times of jobs by the number of jobs during a scheduling period. Equation 2.9 explains the calculations of the mean flow time of a set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$. The $\gamma = \overline{F}$ is often adopted to address scheduling problems in cloud (Remesh et al., 2022), manufacturing (Niu et al., 2012), and supply chain (Selvarajah and Zhang, 2014). In the literature, several objective functions are derived from the flow time, such as the $F_{\max} = \max(F_j) : \forall\ j \in \{1, \ldots, n\}$ based on Equation 2.4 (Baker and Trietsch, 2009, pp. 15-16), the total flow time (Baker and Trietsch, 2009, pp. 16-19). Similar to the makespan $C_{\max}$, the popularity of the flow time in different fields is explained by its direct correlation with the overall efficiency of any scheduling environment. Moreover, early investigations of the average flow time suggested that pursuing the minimization of the $\overline{F}$ yield to indirectly minimize the mean completion time $\overline{C}$ and the mean lateness $\overline{L}$ (Brah and Hunsucker, 1991; Blackstone et al., 1982; Conway et al., 1967).

$$\overline{F} = \sum_{j=1}^{n} F_j/n : \forall\ j \in \{1, \ldots, n\} \tag{2.9}$$

The total tardiness $\gamma = T$:

As the name suggests, the total tardiness is computed by summing up the tardiness values of all jobs during a scheduling period as presented in Equation 2.10. It is an objective value that we usually adopt to address customer satisfaction concerns(Baker and Trietsch, 2009, p. 86). Unfortunately, the popularity of system efficiency concerns such as the makespan led to overlooking the total tardiness concerns in the scientific literature (Neufeld et al., 2016, p. 61; Ribas et al., 2010, p. 1451). Similarly to the average flow time, total tardiness is also associated with several other objective values. For instance, it is common to combine the minimization of total tardiness with the minimization of the total number of penalties during a scheduling period (see. Equation 2.7) to address customer satisfaction concerns. The maximum tardiness, denoted by T_{max}, is another variation of the tardiness-based objective values. It is computed by identifying the maximum tardiness value of a set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ during a scheduling period. It is often used to determine the quality of scheduling policies (Baker and Trietsch, 2009, pp. 21-25).

$$T = \sum_{j=1}^{n} T_j,\ T_j = \max\left(C_j - d_j\right) : \forall\ j \in \{1, \ldots, n\} \tag{2.10}$$

The total number of major setup times $\gamma = MS$:

If a scheduling environment is subject to the family setup times constraint, the total number of family major setup times has a significant impact on its performance. The setup time is a lost configuration time, which is required to prepare a machine before starting to process a job. The total number of major setup times can be computed according to

Equation 2.11. Given a set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ that are grouped into the set of families $f = \{f_g, \ldots, f_{|f|}\} : \forall\ g \in \{1, \ldots, |f|\}$. They must be processed using the set of available machines $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$. The number of major setup times during a scheduling period is increased by one every time a machine M_i finishes processing a $J_j \in f_g$ and starts processing a job $J_k \in f_h$ (Wittrock, 1990; Kut C. So, 1990).

$$MS = \sum_{i=1}^{m} MS_i,\ MS_i = \begin{cases} 1 \text{ if } J_j \in f_g \wedge J_k \in f_h : J_j \prec J_k \\ 0 \text{ Otherweise} \end{cases} \tag{2.11}$$

The total energy consumption $\gamma = E$:

As the name suggests, this objective value is used to seek the minimization of the overall consumed energy. Equation 2.12 depicts a generalized computation of the overall energy consumption in a scheduling environment. It is computed by summing up the average energy consumption $\overline{E}_{i,j}$ of every machine $M_i \in M$ for processing every job $J_j \in J$ during a scheduling period. This objective function has recently been quite popular for solving scheduling problems in cloud environments (Jayanetti et al., 2022; Ghafari et al., 2022; Challita et al., 2017; Pires and Barán, 2015; Mouftah and Kantarci, 2013). For the sake of consistency, we presented a generalized definition of energy consumption in Equation 2.12. Depending on the considered scheduling environment, a machine's energy consumption is an integral of the machine utilization function over time (Beloglazov and Buyya, 2010, p. 759).

In cloud environments, a physical machine's energy consumption profile depends on the current workload on the machine (Beloglazov and Buyya, 2010, p. 759). Many studies presented methods that aim to minimize the number of active machines to reduce energy consumption (Chen et al., 2021; Nahhas et al., 2019a; Varasteh and Goudarzi, 2017). This practice is motivated by several studies, which systematically associated the increase in energy consumption of cloud environments with the increase in the number of machines (Koomey, 2011, p. 9; Koomey, 2008, p. 4). We will refer to this objective value in association with the energy consumption by $min(M_m)$. In manufacturing, the total energy consumption is not well-researched and started emerging as a new objective measure for solving industrial scheduling problems (Neufeld et al., 2023, p. 3; Chou et al., 2020).

$$E = \sum_{i=1}^{m} \sum_{j=1}^{n} \overline{E}_{i,j} * p_{i,j} : \forall\ i \in \{1, \ldots, m\} \wedge \forall\ j \in \{1, \ldots, n\} \tag{2.12}$$

2.1.4 Summary of scheduling

Based on the presented preliminaries of scheduling problems and the presented notation of (Graham et al., 1979), we must account for four main elements while modeling a scheduling problem before designing a solution technique. During the design, development, and evaluation stages of the research artifact, we addressed several single-stage scheduling problems in cloud environments $\langle Q_m \mid prmp,\ M^C \mid E,\ U,\ C_{max}\rangle$ taking into account the minimization of energy consumption while accounting for SLA violations. Similarly, we studied various variations of the multi-stage scheduling problems $\langle HFS_m \mid f_{g,h} \mid C_{max},\ T,\ U\rangle$. The problems are investigated with the objective of minimizing the makespan, the total tardiness, and the number of penalties in a manufacturing environment. For the evaluation, we conducted extended experiments on rather more complex multi-stage scheduling problems in manufacturing, such as the two-stage system to minimize the makespan $\langle HFS_2\ (P_5,\ P_4) \mid f_{g,h} \mid C_{max},\ T\rangle$, a four-stage scheduling environment $\langle HFS_4\ (Q_5, Q_5, P_2, P_2) \mid f_{g,h} \mid C_{max},\ T,\ U,\ MS\rangle$ taking into account multiple objective optimality criteria.

2.2 Simulation

Simulation methods are popular techniques for modeling and investigating different phenomena in complex environments. The process of building a simulation model starts with designing a conceptual model of a considered environment. A conceptual model for simulation sketches the structure of the considered environment, either exactly or subject to some level of abstraction, to produce initial blueprints for a considered environment. Engineers can start building the digital simulation model based on the conceptual model. A simulation model is a digital computer program that emulates the considered environment subject to applied abstraction practices. It is a collection of rules, procedures, equations, or flow diagrams that determine how the modeled system will behave in the future based on its current state (Borshchev and Filippov, 2004).

Based on the nature of the analyzed environment, two simulation paradigms are widely adopted, namely, discrete event and system dynamics or continuous (Angerhofer and Angelides, 2000; Reggelin and Tolujew, 2011). If modeling a considered system requires foundational elements from both paradigms, engineers usually resort to some hybrid simulation frameworks such as discrete rate or mesoscopic simulation methods (Borshchev and Filippov, 2004; Reggelin and Tolujew, 2011). The foundations of system dynamics were initially introduced by Forrester in 1968 under the name "Industrial Dynamics". It was presented to describe and illustrate the dynamic behavior of complex systems, internal communication flows, and their interconnectivity. The concept had been refined as a modeling method and eventually became what we know today as System Dynamics (SD) (Forrester, 1968). The SD is a widely adopted simulation technique characterized by continuous behavior and dynamic change of its elements. Real-world processes in SD are represented as stocks (e.g., of material, knowledge, people, and money), flows between

these stocks, and information that affects the values of the flows (Borshchev and Filippov, 2004). The transformation of the processes into flows requires a high level of abstraction. Hence, processes are represented in a rather aggregated form and support strategic decision-making processes instead of operative ones (Reggelin and Tolujew, 2011). Therefore, such models are often referred to as macroscopic models (Pierreval et al., 2007).

However, simulating complex production or scheduling processes requires high accuracy and granularity, necessitating the adoption of discrete event simulation practices. The fundamentals of Discrete Event Simulation (DES) were first introduced by Gordon in 1961 (Gordon, 1961). DES is a widely utilized approach to model complex manufacturing systems by mapping people, tasks, and resources as objects (Helal et al., 2007; Borshchev and Filippov, 2004). The modeling of a system using the DES approach involves transforming all system elements into active or passive objects. Objects that move in the system, such as jobs, are active or dynamic objects. Passive or static objects are, for instance, resources, queues, or machines that perform operations on the dynamic objects (Krahl, 2008). Unlike SD models, DE models do not change continuously over time; instead, changes in the system's state and its elements are associated with specific events triggered during a well-defined simulation period. Due to their ability to represent workstations, equipment, transportation, and raw material units as individual objects, DE models can achieve a high-level microscopic representation of a considered environment (Pierreval et al., 2007; Reggelin and Tolujew, 2011).

Additionally, DE models are often adopted to visualize and investigate detailed operational workflows of a system in order to collect precise and detailed information (e.g., production planning) (Helal et al., 2007). Hence, discrete event simulation models have often been combined with optimization strategies to address scheduling problems (März et al., 2011). However, modeling such complex and large-scale systems comes with substantial challenges in terms of the high computational power and the extensive modeling effort required to build these models (Reggelin and Tolujew, 2011). Therefore, automation is crucial to operating and maintaining complex simulation models. In this work, we utilize the foundations of the discrete event simulation paradigm to build simulation models and simulate phenomena of discrete nature.

2.3 An overview of conventional solution methods

Given the preliminaries we discussed in previous sections, we may classify scheduling problems into single-stage P_m, multi-stage HFS_m, or job shop Jm (Baker and Trietsch, 2009; Pinedo, 2012). Similarly, since the adopted solution methods strongly depend on the complexity of a studied problem, we may classify them based on the degree of complexity to achieve optimal solutions. The term complexity refers to the amount of computational power necessary to solve an arbitrary problem using some algorithm. An algorithm is a finite collection of procedures and instructions that is assembled into a script to solve a given problem or perform a predefined task (Sipser, 2012). An important area of theoretical computer science is specifically focused on complexity theory. This theory classifies

optimization problems into complexity classes according to the computational power required to solve them.

One of the earliest works in this area was the introduction of the NP-complete class in the early seventies by Cook (1971). This class contains many well-known hard combinatorial optimization problems. Given the current technological stand, an algorithm cannot solve these problems optimally in polynomial time. For some of these problems, the complexity is measured even higher, falling into the NP-hard class (Johnson, 2012). To find the optimal solution for a scheduling problem, we may fully explore the solution space to guarantee optimum since we must compute the quality of every solution given an objective function. Searching the solution space of a problem entirely is usually referred to as complete enumeration. However, even for a small scheduling problem, an enormous number of possible solutions may exist, and that number usually grows exponentially.

For example, Gupta and Stafford (2006) elaborated through a simple example of how the complexity of a scheduling problem can grow. They demonstrated that a scheduling problem with five jobs and five different machines may have $(5!)^5$ number of possible solutions, given some objective function. Their objective is to emphasize how quickly expensive it can be to explore all possible solutions to a small scheduling problem fully. Several years after the introduction of the NP theorem in general (Cook, 1971), Lenstra et al. (1977) investigated and summarized the complexity of scheduling problems in particular. The authors followed the logical flow of Cook (1971) and surveyed classical scheduling problems in pursuit of their categorization. They emphasized known scheduling problems that are found to be NP-Hard. The study also summarized other scheduling problems, which can be solved in polynomial time using some algorithms. Lenstra et al. (1977) reiterated the central role of operational constraints in increasing the complexity of typical scheduling problems.

For instance, the $s_{j,k}$ constraint greatly affects the complexity of the scheduling problem since sequence-dependency leads to high setup times (Maccarthy and Liu, 1993). A practical example is the $\langle 1 \mid s_{j,\,k} \mid C_{max} \rangle$, which is an NP-Hard problem since constraining the minimization of the makespan by sequence dependency leads to an exponential increase in the number of possible solutions (Gupta and Kyparisis, 1987). This complexity is deduced by reducing the single machine problem to the well-known traveling salesman problem, which is NP-hard (Gilmore and Gomory, 1964).

In general, depending on their solution quality and implementation approach, methods for solving scheduling problems can be divided into two main groups as presented in Figure 2.5. We derived this classification based on literature analysis and aligned it with various proposals and analyses presented in (Ribas et al., 2010, p. 1445; Reza Hejazi and Saghafian, 2005; Stützle, 1998, p. 1560).

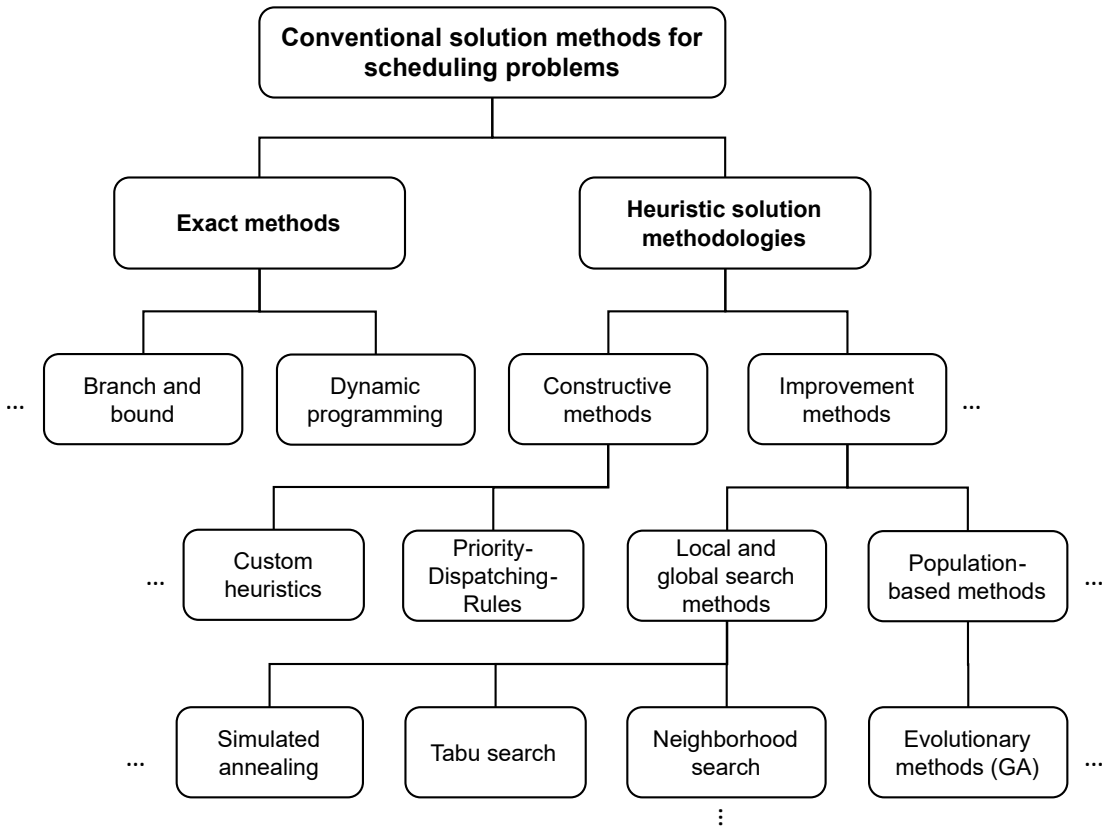


Figure 2.5: Classification of solution methods for scheduling problems extend based on (Ribas, 2010, p. 1445; Reza Hejazi and Saghafian, 2005)

The first group encompasses exact solution methods that pursue optimal solutions or their approximation (Baker and Trietsch, 2009). For instance, Dynamic programming (Held and Karp, 1961) or branch and bound (Kis and Pesch, 2005) are particularly effective for addressing small- to possibly medium-sized scheduling problems. These methods, among the exact approaches, are commonly used for solving scheduling problems. In addition to these methods, several special algorithms can guarantee optimal solutions for solving certain types of problems (Gupta, 1988; Gupta and Kyparisis, 1987; Johnson, 1954). For instance, in 1954, Johnson (1954) contributed an influential paper in the flow shop scheduling research. He investigated the two-stage flow shop scheduling problem $\langle F_2 \ (1, \ 1) \mid s_{j, \ k} \mid C_{max} \rangle$. He introduced an exact algorithm capable of minimizing the makespan with sequence-dependent setup time in polynomial time for a two-stage flow shop scheduling problem with a single machine at every stage. Although the size of the considered problem is still quite small, his algorithm has been combined to solve larger problems in the flow shop literature, for instance, in (Gupta, 1988).

Kis and Pesch (2005), have undertaken a thorough review of exact methods for solving multi-stage HFS_m scheduling problems and focused on the branch-and-bound solution method. Although these solution techniques guarantee optimal or bounded optimal solutions for a given scheduling problem, they become computationally very expensive as problem complexity increases. This conundrum has always been approached by excessive

abstraction in modeling a problem. That, in turn, leads to an oversimplification of real-world environments. As a result, a transfer gap between scheduling theory and industrial practices is evident. This problem has been well-recognized since the early nineties by a study presented by Maccarthy and Liu (1993, p. 70).

In this thesis, we deal with complex single- and multi-stage scheduling problems. Therefore, we focus on the second group, which encompasses heuristic solution methods as presented in Figure 2.5. Although heuristic methods may not guarantee optimal solutions, they provide viable and efficiently implemented solutions for large-scale scheduling problems. Under heuristic methods, we categorize often adopted solution techniques for solving scheduling problems into constructive and improvement methods. First, we will start with constructive methods. Then, we will provide an overview of common improvement techniques before focusing on methods related to this work.

2.3.1 Constructive heuristic methods

As the name suggests, these methods are designed to construct a solution for a scheduling problem following certain procedural logic (Ribas et al., 2010, p. 1445; Reza Hejazi and Saghafian, 2005, p. 2906; Stützle, 1998, p. 1560). It means that after a solution for a scheduling problem is constructed, no further improvements are researched. In practice, scheduling problems in cloud (P_m, Q_m) and manufacturing HFS_m environments are predominantly approached by heuristic solution methods (Ghafari et al., 2022, p. 1045; Pires and Barán, 2015, p. 164; Ruiz and Vázquez-Rodríguez, 2010, p. 21; Ribas et al., 2010, p. 1452).

Historically, the scheduling research field evolved during the third industrial revolution and mass production. Therefore, the majority of simple constructive heuristics are presented in the field of manufacturing and later on adopted or modified for addressing scheduling problems in cloud environments. Consequently, a very important sub-class of constructive heuristic methods encompasses the so-called Priority Dispatching Rules (PDRs) in manufacturing environments. A priority dispatching rule is usually designed to prioritize released jobs by constructing a sequence that defines their importance. The significance of a job is computed based on some job and/or capacity data that is associated with some pursued objective values. In essence, most of the PRDs act as a ranking mechanism ranging from simple to complex (Blackstone et al., 1982, p. 27).

Already by the early eighties, dozens of PDRs were developed and adopted for addressing scheduling concerns in the industry (Blackstone et al., 1982, p. 27; Hunsucker and Shah, 1994, p. 104). One of the earliest surveys of PDRs is presented by Blackstone et al. (1982), in which the authors investigated and discussed 34 PDRs that are already utilized in industry (Blackstone et al., 1982, pp. 33-43). The authors emphasized the intuitive and simplistic nature of developing such constructive heuristics. They added in their final remarks that it is impossible to identify the best rule for all circumstances (Blackstone et al., 1982, pp. 27-28). PDR techniques are efficient when we pursue the optimization of a single objective in solving a scheduling problem.

Algorithm 1 Example of Earliest Due Date for P_m

```
    /* Equation 2.6 - We sort ascending in terms of due date d_j */
 1: procedure EDD (J, M)
 2:     Z_solution ← ∅
 3:     J_sorted ← sortAscendingByDueDate (J, index = d_j)
 4:     M_sorted ← sortAscendingByWorkload (M, index = l_i)
 5:     while J_sorted ≠ ∅ do
 6:         J_j ← getFirstJob (J_sorted)
 7:         M_i ← getFirstMachine (M_sorted)
 8:         Z_solution ← constructSolution (J_j, M_i)
 9:         J_sorted ← remove (J_sorted, J_j)
10:         M ← updateWorkload (M, M_i, p_j)
11:         M_sorted ← SortAscendingByWorkload (M, index = l_i)
12:     end while
13:     return Z_solution
14: end procedure
```

For instance, the Earliest Due Date (EDD) rule is a simple constructive heuristic that sorts a set of released jobs for scheduling in an ascending order according to their due date to construct a scheduling sequence taking into account the available machines. It implies that the job with the earliest due date is dispatched first for processing. This rule is repeated until the last job is processed. Algorithm 1 depicts a pseudocode for the EDD algorithm for solving the single-stage scheduling problem with parallel machines. We may have a set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$, which we want to schedule for processing. In our environment exists a set of machines $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$. Let their workload be denoted by a set $l = \{l_i, \ldots, l_m\} : \forall\ i \in \{1, \ldots, m\}$. In the case of parallel machines, machines are also sorted in terms of their workload in ascending order (Algorithm 1, Lines 4 and 11).

The EDD rule is often applied to address scheduling concerns subject to the minimization of the total tardiness and/or the total number of penalties during a scheduling period. In this algorithm, we assume that machines have buffer capacities and jobs await processing over time according to the obtained solution. For the sake of simplification and generality, we omit some sorting logic and other simple operations to sustain a simple pseudocode presentation. For instance, we assume that there are some sorting functions that we can directly use without explaining their logical flow, such as sorting ($sortAscendingByDueDate$), get functions ($getFirstJob$), ($getFirstMachine$), update function ($updateWorkload$), or mapping function ($constructSolution$). We will follow the same conventions when presenting all algorithms in the thesis.

Another well-known and often adopted PDR is the Shortest Processing Time (SPT) heuristic. The algorithm was applied to deal with a machine reparation problem in the fifties by Phipps (1956, p. 76). Many works followed investigating the performance of the SPT rule in solving scheduling problems (Schrage and Miller, 1966; Schrage, 1968).

Algorithm 2 Example of Shortest Processing Time for P_m

```
    /* See. Subsection 2.1.1 - We sort ascending in terms of processing time p_j */
 1: procedure SPT (J, M)
 2:     Z_solution ← ∅
 3:     J_sorted ← sortAscendingByProcessingTime (J, index = p_j)
 4:     M_sorted ← sortAscendingByWorkload (M, index = l_i)
 5:     while J_sorted ≠ ∅ do
 6:         J_j ← getFirstJob (J_sorted)
 7:         M_i ← getFirstMachine (M_sorted)
 8:         Z_solution ← constructSolution (J_j, l_i)
 9:         J_sorted ← remove (J_sorted, J_j)
10:         M ← updateWorkload (M, M_i, p_j)
11:         M_sorted ← SortAscendingByWorkload (M, index = l_i)
12:     end while
13:     return Z_solution
14: end procedure
```

The SPT rule ranking system is mainly based on the job processing time. Algorithm 2 shows an example of the SPT for solving a scheduling problem with parallel machines P_m. As shown in Algorithm 2, given a set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$, the SPT heuristic sorts jobs according to their processing time p_j in ascending order before dispatching (Algorithm 2, line 3). Given a set of machines $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$, let their workload be denoted by a set $l = \{l_i, \ldots, l_m\} : \forall\ i \in \{1, \ldots, m\}$. Every machine M_i is associated with its current workload l_i. Machines are sorted in terms of their workload in ascending order before scheduling released jobs (Algorithm 2, lines 4 and 11).

The SPT rule shows significant performance for minimizing the flow time-related objective values such as $\overline{F}$ or $\sum F$, especially in the flow shop scheduling environments (Baker and Trietsch, 2009, p. 58). It can also be effective for minimizing the C_{max} (Baker and Trietsch, 2009, p. 234) as we could see in the Johnson (1954) algorithm. The SPT is interchangeably called Min-Min in many scientific works addressing scheduling problems in cloud environments. The first Min stands for the minimal job, and the second Min stands for the minimally loaded available machine.

Similarly, the Longest Processing Time (LPT) is another popular PDR for addressing scheduling concerns in practice. Algorithm 3 presents the logical flow of the SPT heuristic for the parallel machines problem. The ranking of jobs is also based on the processing time. In contrast to the SPT, the jobs are sorted in descending order (Algorithm 3, line 3) before being scheduled while following a similar logical flow of the previous PDRs. The LPT is referred to as Max-Min in many scientific works addressing scheduling problems in cloud environments. The Max refers to the maximum length of overall jobs and the Min points to the machine that is minimally loaded.

Algorithm 3 Example of Longest Processing Time for P_m

```
/* Equation 2.3 - We sort descending in terms of processing time p_j */
1: procedure LPT (J, M)
2:     Z_solution ← ∅
3:     J_sorted ← sortDescendingByProcessingTime (J, index = p_j)
4:     M_sorted ← sortAscendingByWorkload (M, index = l_i)
5:     while J_sorted ≠ ∅ do
6:         J_j ← getFirstJob (J_sorted)
7:         M_i ← getFirstMachine (M_sorted)
8:         Z_solution ← constructSolution (J_j, M_i)
9:         J_sorted ← remove (J_sorted, J_j)
10:        M ← updateWorkload (M, M_i, p_j)
11:        M_sorted ← SortAscendingByWorkload (M, index = l_i)
12:    end while
13:    return Z_solution
14: end procedure
```

After almost thirty years of application in industry, the most popular priority dispatching rules are once again investigated for addressing multi-stage problems HFS_m (Hunsucker and Shah, 1994). In their investigation, Hunsucker and Shah (1994) systematically studied the performance of the SPT, LPT, and several other PDRs. They focused on testing the performance of previous PDRs for solving the $\langle HFS_m \mid \mid C_{max}, F_{max}, \overline{F} \rangle$. For instance, they analyzed the Shortest Remaining Processing Time (SRPT) and the Longest Remaining Processing Time (LRPT) in addition to the First-In-First-Out (FIFO) and Last-In-First-Out (LIFO). The SRPT and LRPT rules are also based on the processing time of released jobs during a scheduling period similar to the SPT and LPT, respectively (Baker and Trietsch, 2009, p. 168). However, we consider not only the processing time of a job in the first processing stage but also the processing time of all upcoming processing stages.

The makespan, mean flow time, and maximum flow time are common objectives in HFS_m scheduling environments. Taking into account different workload levels and system configurations, the findings showed that the SPT PDR is very effective and outperforms all rules for addressing the HFS_m scheduling problems subject to the minimization of the makespan (Hunsucker and Shah, 1994, pp. 110-112). Obviously, the $\langle HFS_m \mid \mid C_{max} \rangle$ scheduling problems are significantly simpler given a single objective and no constraints. A rather more recent study on the performance of PDRs with due date characteristics is presented by (Đurasević and Jakobović, 2020) for addressing the $\langle Q_m \mid \mid C_{max} \rangle$ subject to the minimization of the makespan. The authors compared various rules against automatically developed rules for solving small problem instances with 11 jobs and three parallel machines. The computational results suggest that the automatically generated rules outperform standard PDRs in the literature. A couple of years later, Almeida et al. (2022) revisited the performance of PDRs for addressing flow shop

problems $\langle Fm \mid \mid \sum F,\ U,\ M_{utilization} \rangle$. The maximization of the $M_{utilization}$ objective value is usually achieved through the minimization of the machine's idle time. The authors surveyed the performance of EDD, SPT, LPT, FIFO, LIFO, and some other rules based on the Number of Operations (NOP) or Ratio of Criticality (RC). The authors conducted experiments on randomly generated problem instances. The computational results suggest that most PRDs are, on average, 14 % better than the FIFO rule (Almeida et al., 2022, p. 264), which is often used in industry. The results indicate that the EDD and SPT rules, once again, are superior in terms of minimizing the U and the average machine utilization $M_{utilization}$ in flow shop environments.

Around the same time Ashwin et al. (2022) presented a comprehensive analysis of PDRs and their possible combination for solving the $\langle Jm \mid \mid C_{max},\ T_{max},\ T,\ F \rangle$. The authors focused in their analysis on flow time- and tardiness-based objective values. They evaluated the performance of the EDD, Minimum Slack Time (MST), SPT, LPT, FIFO, and four proposed variations of their combination. The slack time is computed by subtracting the earliest possible finish time of a job from its due date (Baker and Trietsch, 2009, p. 353). The authors evaluated the concepts by solving small problem instances comprising 6 and 6 machines as well as ten jobs and ten machines. The computational results showed that the proposed combinations yield better performance in minimizing the objective values (Ashwin et al., 2022, p. 343). The EDD rule remains effective for addressing tardiness-related objective measures. The SPT proved outperformance in terms of minimizing the mean flow time for solving the smaller problem instance.

In contrast to previous studies, Spanos et al. (2022) analyzed various PDRs to include them in the development of a decision support system for a small company. The authors surveyed experts to finalize the included list of rules that included EDD, SPT, LPT, MST, Shortest Setup Time (SST) , and Pre-Subcontractor First (PSF). The PSF rule prioritizes certain customers compared to others. The authors suggested hierarchical interactive scheduling, where experts select among available rules and objective values to construct a solution for every machine given a partial set of jobs. The computational results demonstrated that the SPT, once again, is superior for minimizing the $\overline{F}$, while the EDD with weights delivers the best results for minimizing the weighted T_{max}. As expected, the authors recommended the combination of various PDRs to achieve the best results when the problem is subject to multiple objective values (Spanos et al., 2022, p. 13).

Weng et al. (2022) investigated a rather complex job shop problem with multiple processing stages. The authors focused on due-date-related objective measures such as the minimization of total tardiness and earliness. They proposed several modified PDRs that are based on the classical EDD, MST, and FIFO PDRs and relied on the decomposition of the problem to apply the rules. The computational results demonstrated that the proposed rules are effective for minimizing the considered objective values. The authors argue that the computational results of the proposed rules outperform several improvement methods for solving randomly generated problem instances that involved up to 96 jobs (Weng et al., 2022, pp. 12-14). A similar proposal was presented by Kasper et al. (2023), in which the authors presented a system-based dispatching method that relies on modified variations

of standard dispatching rules and system information. The author presented an extensive analysis of the method for solving flow shop scheduling problems to minimize the average tardiness and percentage of late jobs. The reported computational results demonstrated an outperformance of the presented method compared to standard PDRs such as SPT, EDD, and MST.

The vast adoption of PDRs for addressing scheduling problems in the manufacturing environment makes it impossible to survey all rules. Their popularity extends beyond manufacturing environments. Their adoption is surely notable for addressing scheduling problems in cloud environments. However, we may notice slight differences in the naming of these PDRs. For instance, (Sindhu and Mukherjee, 2011) presented a brief analysis of the performance of the Shortest-Job-First, the Longest-Job-First, and the First-Come-First-Served, which are equivalent to the SPT, LPT, and FIFO, respectively. To maintain consistency, we will use the conventional naming of PDRs in scheduling. The analysis is conducted to solve a simple parallel machine scheduling problem to minimize the makespan $\langle P_m \mid \mid C_{max} \rangle$. The authors relied on simulation techniques to investigate the problem. The simulation results suggest that the SPT and the LPT outperform the FIFO rule, especially when the number of jobs to be scheduled is increased. However, the conducted analysis is subject to a small number of jobs.

Similarly, Murad et al. (2021) investigated the performance of several PDRs for addressing the single-stage scheduling problem $\langle Q_m \mid \mid C_{max}, M_{utilization}, F_{related} \rangle$. The authors compared the performance of the SPT and LPT. They also adopted a simple ML classifier to solve the problem. The computational results demonstrated that the SPT rule effectively schedules jobs subject to the considered objective values. The experimental setup included only five machines. Similarly, Murad et al. (2023) proposed a new PDR for scheduling jobs in parallel machines cloud environments considering the $\langle Q_m \mid \mid C_{max},\ F,\ T \rangle$. Given minimizing the makespan, flow time, and total tardiness, the authors proposed a rule that relies on the release and due date of the job. Finally, this rule combines various SPT and EDD ranking mechanisms for scheduling jobs on machines. The paper presented computational results that compare the performance of the new rule against known PDRs such as the EDD, SPT, LPT, FIFO, Min-Min, and Max-Min. The Min-Min and Max-Min rules are some variation of the SPT and the SPT, respectively (Algorithm 2 and Algorithm 3), that are used for parallel machine problems. The proposed rule achieved a similar performance to the SPT in terms of minimizing the flow time. The experiments demonstrated that the new rule finds a trade-off solution that minimizes the total tardiness and the flow time Murad et al. (2023, p. 174).

For addressing a similar problem $\langle Q_m \mid \mid C_{max}, \rangle$, Bandaranayake et al. (2020) proposed a new PDR called Total Resource Execution Time Aware Algorithm (TRETA) for scheduling jobs in cloud environments. In addition to the minimization of the makespan, the authors also reported results on the throughput and workload balance between machines. The proposed rule is based on the processing time of a job, in which the job is scheduled on the fastest and first available machine. The authors compared the performance of TRETA against the FIFO, SPT, Min-Min, and Max-Min priority dispatching

rules. To investigate practicality, the authors relied on the NASA benchmark for generating jobs in a simulation model. The simulation results demonstrated the superiority of TREAT against all PDRs for solving several scheduling problems with different numbers of jobs that must be scheduled.

Shin et al. (2022) addressed a rather more complex scheduling problem in which jobs must undergo several processing steps $\langle P_m \mid prec \mid C_j,\ C_{max} \rangle$. In essence, a job $J_j \in J$ consists of a set of operations $O_j = \left\{O_{o,j}, \ldots, O_{|O|,j}\right\} : \forall\ o \in \{1, \ldots, |O|\}$ forming a workflow that must be scheduled for processing in a cloud environment. The authors proposed a new rule combining the Maximum Children (MC) and the weighted job completion time $(w_j * C_j)$, where w_j denotes a weight assigned to a job J_j indicating its importance. The MC rule sorts jobs in descending order in terms of their number of operations. The authors compared the performance of the presented rule against the SPT, FIFO, and MC for scheduling ten jobs on 20 machines. The experimental results showed that the proposed rule outperforms individual rules for solving the problem.

In Subsection 2.1.3, we discussed the minimization of energy consumption in scheduling as an emerging objective measure. Especially in cloud environments, minimizing energy consumption became crucial since some regulations on CO_2 emission have been posed in many countries. Therefore, many research efforts have been dedicated to proposing scheduling methods that take into account energy efficiency. Akhter and Othman (2016) reviewed and thoroughly discussed open challenges in reducing energy consumption in cloud environments. The authors stressed the role of energy-aware scheduling methods in reducing energy consumption in large cloud environments. Several years later, Ghafari et al. (2022) surveyed the past ten years of research and development efforts in the field of scheduling in cloud environments, focusing on energy-ware solutions. The authors stressed in their findings the dominance of heuristic and improvement methods for proposing energy-efficient scheduling methods (Ghafari et al., 2022, p. 1045). They also studied other popular objective values in addition to minimizing the energy consumption E. Their summary suggests that around 73 % of proposed methods also pursue minimizing the average completion time $\overline{C}$, and some 32 % concentrate on minimizing the makespan C_{max}. As a result, other objective values, such as penalties U or violation of SLAs, are somewhat overlooked.

Many energy-aware solutions in cloud environments are presented to address the virtual machine placement problem. We will revisit our discussion regarding the machine capacity constraint and energy objective value in Subsection 2.1.2. Suppose we have a set of jobs (virtual machines, J^R) that must be scheduled on a set of machines (computing servers, M^C). The set of jobs is denoted by $J^R = \left\{J_j^R, \ldots, J_n^R\right\} : \forall\ j \in \{1, \ldots, n\}$. Let the set of available parallel machines denotes $M^C = \left\{M_i^C, \ldots, M_m^C\right\} : \forall\ i \in \{1, \ldots, m\}$. Many similar problems are reduced to bin-packing problems (Akhter and Othman, 2016, p. 1171). In a parallel machine scheduling environment, depending on the considered objective function, we may have to deal only with the allocation dimension of a scheduling problem (Baker and Trietsch, 2009, p. 221). In this example, a job $J_j^R \in J^R$ may be allocated to a machine $M_i^C \in M^C$, only if the machine can satisfy all its resources

requirements ($\vdash \Upsilon$, Equation 2.1).

For instance, a job J_j^R can be scheduled on a server only if there is enough compute R_{CPU}, memory R_{Ram}, and storage R_{SSD} to process it. Algorithm 4 is a generalized heuristic method, which we may use to allocate jobs to the machines given the capacity constraint considering the minimization of the overall energy consumption. The algorithm is a modified version of the First-Fit-Decreasing (FFD) algorithm (Yue, 1991), taking into account the energy consumption. The FFD logic is based on the LPT priority dispatching rule, in which a set of jobs are sorted in descending order in terms of their capacity requirements R_r (Algorithm 4, Line 5). Here, we assume that the energy consumption to process a job $J_j^R \in J^R$ is strongly dependent on some resource requirement R_r, which we use as an index for sorting (e.g., R_{CPU}).

After initialization, the algorithm loops over the sorted list of jobs starting in (Algorithm 4, Line 6). Then, the first job is selected for scheduling, and a set of requirements is extracted using some simple function. Finally, the heuristic procedure loops over all available machines starting in (Algorithm 4, Line 12). Then, we check if the current selected machine M_i^C has enough capacities to process the job J_j^R (Algorithm 4, Lines 13-20). Depending on the results, we estimate the incurred energy consumption for processing the job using some simple procedures and assume that it is, for the moment, the minimal one (Algorithm 4, Lines 21-27). We repeat this process for all available machines until. Finally, if we find a suitable machine, we will update our intermediate schedule and update the workload of our machines (Algorithm 4, Lines 29-32). Otherwise, we add this job to unscheduled jobs and start investigating the next one back-in (Algorithm 4, Lines 6). After looping over all jobs, we get our solution, which is a map that schedules jobs to corresponding machines. We may also get a set of unscheduled jobs that must be planned later.

The FFD selects the first job in the sorted job list and attempts to place it on the first machine. In our example, we iterate over all parallel machines to find a machine with enough capacity to satisfy the requirements of the job and consume the least energy to process it (Algorithm 4, Lines 14-28). The logical flow of the classical FFD would stop after finding the first possible fit (machine) for the job that can satisfy its requirements. If we add a break statement in (Algorithm 4, after Line 24) to stop the iteration, we have a classical FFD. Of course, we still need to update the workload of the selected machine and the intermediate solution.

Algorithm 4 Example Energy-Aware Algorithm Q_m

```
    /* Equation 2.1 - We sort according to the first requirement in line 5 */
 1: procedure EnergyAware (J^R, M^C)
 2:     Z_solution ← ∅
 3:     J_notScheduled ← ∅                          ▷ A partial set of jobs that may have to wait
 4:     R ← getJobRequirments(J^R)                  ▷ Extract the set of job requirements
 5:     J^R_sorted ← sortDescendingByResourceRequirement(J^R, index = R_r)
 6:     while J^R_sorted ≠ ∅ do
 7:         Ē_j^min ← MAX                           ▷ The minimal energy consumption of a job as max
 8:         M^C_{i,j} ← NULL                        ▷ The job is not scheduled on any machine yet
 9:         satisfied ← False                       ▷ Boolean to check requirements are met
10:         J^R_j ← getFirstJob (J^R_sorted)
11:         R ← getJobRequirments(J^R_j)            ▷ Extract the set of job requirements
12:         for each M^C_i ∈ M^C do
13:             C ← getMachineCapacities(M^C_i)
14:             for each (R_r ∈ R) and (C_c ∈ C) do
15:                 if R_r ≤ C_c then               ▷ Machine capacity satisfies job's requirement?
16:                     satisfied ← True
17:                 else
18:                     satisfied ← False
19:                 end if
20:             end for
21:             if satisfied == Ture then
22:                 Ē_j ← estimatePowerConsumption(J^R_j, M^C_i)
23:                 if Ē_j < Ē_j^min then
24:                     M^C_{i,j} ← M^C_i           ▷ Pick the machine with lower energy
25:                     Ē_j^min ← Ē_j               ▷ update the lowest energy
26:                 end if
27:             end if
28:         end for
29:         if M^C_{i,j} ≠ NULL then
30:             Z_solution ← constructSolution (J^R_j, M^C_{i,j})
31:             J^R_sorted ← remove (J^R_sorted, J^R_j)
32:             M^C ← updateWorkload (M^C, M^C_{i,j}, J^R_j)
33:         else                                    ▷ No machine can satisfy the requirements of a job
34:             J_notScheduled ← addNotScheduledJobs(J_notScheduled, J^R_j)
35:             J^R_sorted ← remove (J^R_sorted, J^R_j)
36:         end if
37:     end while
38:     return Z_solution, J_notScheduled
39: end procedure
```

The logical design of the FFD heuristic method aims mainly at minimizing the number of machines that are actively used. Similarly, the Best-Fit-Decreasing (BFD) heuristic method schedules a job on the most loaded machine that can satisfy its requirements (Moges and Abebe, 2019, p. 4). It implies that we may also sort all machines (including idle ones) in terms of their available capacities in descending order before starting to schedule jobs. Then, we break our loop that searches for a suitable machine in (Algorithm 4, at line 24) directly after we find the fullest machine that meets the requirements of the job. Finally, we need to update the workload of the selected machine and the intermediate solution. In contrast, the Worst-Fit-Decreasing (WFD) logical design is opposite to the BFD. A job is scheduled on the least loaded machine that meets the requirements (Moges and Abebe, 2019, p. 4). Back to our example, we may sort jobs similarly and then sort available machines in ascending order in terms of their available capacity before we start scheduling (Algorithm 4, after line 5). Then, we break our search similarly in (Algorithm 4, after line 24).

These simple constructive heuristic methods are often adopted with various modifications to address scheduling problems in cloud environments. For instance Beloglazov et al. (2012a) presented one of the earliest and most influential contributions among energy-ware heuristic methods for scheduling in cloud environments. The authors addressed a single-stage scheduling problem with machine capacity constraints $\langle Q_m \mid M^C \mid U,\ E \rangle$. The presented heuristic methods aim to minimize energy consumption and the number of penalties due to SLA violations. Here, it is important to discuss that the M^C can sometimes be considered as a soft constraint. It implies that jobs may be scheduled on machines that may not fully meet their requirements at a later point in time during a scheduling period. In a cloud scheduling environment, this phenomenon is widely associated with the notion of overloading or "over-subscription" (Beloglazov et al., 2012b, 1401). The presented methods by Beloglazov et al. (2012a) are inspired by a modification of the FFD, taking into account energy consumption and a rescheduling method to avoid penalties. The authors relied on the CloudSim simulation package to model a cloud environment to conduct the evaluation (Buyya et al., 2009). They named the presented heuristic method Power Aware Best Fit Decreasing (PABFD). The authors compared the PABFD heuristic methods against a random scheduling policy. Simulation results suggest that the presented methods significantly reduced the overall energy consumption of the considered cloud environment (Beloglazov et al., 2012a, p. 762). The authors followed up on their previous research and presented an extended analysis of the presented heuristic methods focusing on the rescheduling processes and its potential to reduce the number of penalties imposed by violation of SLAs (Beloglazov et al., 2012b). The CloudSim and the CloudSim Plus (Silva Filho et al., 2017) simulation packages will discussed in the implementation overview section in the evaluation chapter (cf. Section 4.3).

After several years, Moges and Abebe (2019) revisited the single-stage problem $\langle Q_m \mid M^C \mid U,\ E \rangle$ that is considered by Beloglazov et al. (2012b). They proposed several energy-aware heuristic variations that are also inspired by the FFD (Yue, 1991) and the proposed method by Beloglazov et al. (2012a). They named the heuristic methods as

follows: the Modified Best-Fit Decreasing (MBFD), the Medium-Fit Power Efficient Decreasing (MFED), the Power Efficient First-Fit Decreasing (PEFFD), and the Power Efficient Best-Fit Decreasing (PEBFD). The authors relied on simulation methods to evaluate the performance of the presented algorithms using the CloudSim Plus simulation package (Silva Filho et al., 2017). Moges and Abebe (2019) relied on the PlanetLab benchmark (Park and Pai, 2006), which is widely used for evaluating scheduling heuristic methods in cloud environments (Beloglazov et al., 2012a,b). The computational results over multiple setups suggest that the proposed PEFFD and PEBFD slightly outperform the previous energy-aware heuristic for minimizing energy consumption and SLA violations. Several of the previously discussed heuristics are included in the heuristic library component of the research artifact.

Sun et al. (2024) addressed single-stage workflow scheduling problems in a cloud environment in which a job encompasses multiple operations. The authors considered the minimization of energy consumption and the number of penalties $\langle Q_m \mid prec \mid E,\ U \rangle$. They approached the problem by developing a constructive energy-ware heuristic that considers the energy consumption of machines. The computational results show that the heuristic method outperforms known PDRs in the literature, such as random, round-robin, and ROSA. ROSA heuristic is an EDD-based rule that schedules jobs taking into account due-date related objective measures (Sun et al., 2024, p. 177). The experiments are conducted on known benchmarks in literature with eight machines available for processing jobs.

Evidently, PDRs are problem-specific constructive heuristic methods that deliver reasonable solutions for rather simple problems. The complexity of a scheduling problem substantially increases if we consider operational constraints (Romero-Silva et al., 2022, p. 2; Gupta and Stafford, 2006, p. 701; Wittrock, 1990, p. 331) or conflicting objective values (Neufeld et al., 2023, p. 2; Gupta and Stafford, 2006, p. 701; Wittrock, 1990, p. 331). Complex PDRs, which might construct solutions subject to multiple objective values, are usually associated with high development effort (Ross, 2005, p. 531). To develop such techniques to address specific problems, We require appropriate algorithmic and field knowledge and close communication with domain experts (Ross, 2005, p. 531). Therefore, conventional improvement techniques are quite popular for addressing complex scheduling problems that are subject to multiple objective concerns.

2.3.2 Improvement methods

Improvement methods are conceptually designed with optimization in mind. In the context of scheduling problems, improvement methods start with an initial solution and then seek to refine that solution iteratively to minimize or maximize some objective values. The refinement process is usually based on conducting some changes to the initial solution in pursuit of a better one (Pinedo, 2012, p. 382). Local search algorithms are, for instance, popular improvement methods that are capable of systematically investigating and optimizing an initial solution based on well-defined modification mechanisms. The modification mechanisms must be carefully defined and impartially applied to all investi-

gated solutions (Pirlot, 1996). However, as the name suggests, they are often applied to find optimized solutions in a specific region "neighborhood" of the solution space (Pirlot, 1996, p. 494). In the context of scheduling, two solution candidates can be considered neighbors if a change to one schedule generates the other neighbor's schedule (Pinedo, 2012, p. 382; Pirlot, 1996, p. 494). The changes are performed iteratively on the explored solutions until no improvement can be achieved or some termination condition is satisfied. The modification procedure, which is applied to find the next solution, defines the computational complexity of the algorithm because it determines the size of the explored neighborhood in the solution space Orlin et al. (2004).

The literature contains an enormous number of improvement methods that address different variations of scheduling problems. Hence, we will provide a general overview of conventional improvement methods and their adoption for solving single-stage and multi-stage scheduling problems. The preview is structured based on the problem's complexity, from simplest to most complicated.

For instance, Wittrock (1990) investigated identical parallel machines scheduling problem $\langle P_m \mid f_{g,h} \mid C_{max} \rangle$. The considered environment was solved subject to the family-dependent major and minor setup times constraint aiming at minimizing the makespan. The author briefly discussed the NP-hard complexity of the problem. Hence, an improvement heuristic method was presented to solve the problem. The proposed improvement heuristic consists of three main components. The first component computes the upper and lower bounds on the makespan found so far. The mean of the two bounds is used to find a preferred makespan, which is then passed to the allocation component. The allocation component looks for possible allocations of families to machines and constructs schedules whose makespans are less than the currently calculated average bound. Finally, the third component executes the allocation proposed by the second component. The computational results presented were within three percent of a proven lower bound for optimal makespan, demonstrating the potential of the proposed heuristics to achieve near-optimal schedules.

However, the author made a number of assumptions when presenting the problem definition, such as not taking into account the due date of jobs. It was also assumed that the sequence of production jobs belonging to a single family is trivial as long as they are processed consecutively on the same machine (Wittrock, 1990). These assumptions were made to simplify the sequencing part of the problem and minimize its impact on the overall results by concentrating on the allocation component.

Fiew years earlier, Gupta (1988) addressed an extension of the identical parallel machines with a single machine available on the second stage $\langle HFS_2(P_m, 1) \mid \mid C_{max} \rangle$. The problem was investigated with the intention of minimizing the makespan. A single machine in the second processing stage results in a hybrid flow shop scheduling environment. Despite the fact that both authors intended to minimize the makespan, the form of the problem considered by Gupta is different from the previous problem since no sequence dependency is taken into account. The author proved the complexity of the problem as NP-Complete and justified his proposal of an improvement method to solve it.

Gupta (1988, p. 360) suggested splitting the allocation part of the problem from the sequencing one to produce the solution. Based on this premise, the author assumed a two-stage flow shop scheduling problem that must be solved to minimize the makespan $\langle F(1,1) \mid \mid C_{max} \rangle$. The two-stage flow shop problem is a well-known variation of the flow shop problem, especially after Johnson (1954) proposed an exact algorithm that guarantees optimal solution. Hence, Gupta (1988) utilized the Johnson algorithm to solve the sequencing part of the scheduling problem and proposed an improvement method to solve the allocation part of the problem. The rationale of the proposed method was based on minimizing the overall idle time of the single machine in the second stage and the number of active machines in the first stage. The reported results demonstrated that the proposed heuristic method achieves near-optimal makespan to solve the randomly generated problem instances. The analysis was limited to only two machines in the first stage, reasoning that the heuristic would perform worst due to the machines' unavailability.

A couple of years after these investigations, Voß (1993) revisited the discussion and introduced a Tabu search implementation to address the complicated form of the problem $\langle HFS_2(P_m,1) \mid s_{j,k} \mid C_{max} \rangle$. The author considered the sequence-dependent setup time constraint, which significantly contributes to a higher complexity if the objective function includes the minimization of the makespan. Voß (1993) relied on Gupta's method to produce an initial solution, which is then iteratively refined using Tabu search to minimize the makespan. In his work, Voß (1993) proposed a single-move modification strategy, which swaps the position of two consecutive jobs to obtain the new makespan. The implemented tabu search yielded improved solutions compared to the initially generated ones. The empirical results showed that the presented method achieves a superior makespan compared to the presented method by Gupta (1988), which is used as a baseline. Voß (1993) closed his paper by emphasizing the importance of investigating practical constraints and their significant role in real scheduling environments.

Several years after Voß proposal, Li (1997) presented an improvement method to address two-stage hybrid flow shop scheduling problems. Li (1997) considered, in fact, an opposite variation of the problem with respect to machine environment, in which the first stage contains a single machine and the second stage comprises a set of identical parallel machines $\langle HFS_2(1,P_m) \mid f_{g,h} \mid C_{max} \rangle$. The author investigated the problem, which was subject to family major and minor setup time constraints. Similarly, Li (1997) justified his proposal of improvement method by the NP-completeness of the problem, which can be reduced to the problem addressed by Gupta (1988). The rationale of the proposed method was also based on solving the allocation and the sequencing parts of the problem independently. The authors presented two forward and backward heuristics to deal with the allocation problem. They relied on three priority dispatching rules to solve the sequencing part of the problem in the first processing stage with a single machine. Li (1997) concludes by presenting the results of the two proposed heuristics with the rules and compares their performance with existing PDRs in the literature. In all previously discussed approaches and improvement methods, neither total tardiness nor penalties were considered.

Although local search algorithms often yield good solutions with reasonable computational efforts, they are bound to search only in the neighborhood of some starting solution. Hence, they may likely find the optimal solution in this region, which, however, may not be the global optimal solution for a given problem (Ross, 2005, pp. 529-530). Consequently, many potential feasible solutions that might lead to finding the global optimum are disregarded Voudouris and Tsang (2003). The problem of local optima has been a widely known limitation of these techniques, which compelled many academics to investigate more generalized methods. As a result, numerous advanced optimization improvement methods were proposed under the umbrella of metaheuristic methods Glover and Kochenberger (2003). In general, metaheuristic methods are supervised local search methods (Ross, 2005, pp. 529). Their rationale design incorporates a local search method and an overarching control mechanism. The local search algorithm method is guided and regulated by the controlling mechanism Ross (2005). The ground premise of designing a controlling mechanism is to encourage a local search algorithm, leaving an investigated region of the solution space to explore improvement in another one. However, leaving the explored region of the solution space is conditioned by a reasonable assumption that the local optimum can be mitigated (Crainic and Toulouse, 2003).

For instance, Simulated Annealing (SA) is a widely adopted metaheuristic method for solving scheduling problems (Aurich et al., 2016; Mirsanei et al., 2011; Allaoui and Artiba, 2004). SA was proposed by Kirkpatrick et al. (1983) and empirically evaluated for solving several combinatorial optimization problems. Kirkpatrick et al. (1983) adopted the annealing process of solid physical substances to develop the rationale of his proposed guiding strategy. The annealing process starts by melting down a solid physical substance of some form and then cooling it down gradually to achieve the desired form (Kirkpatrick et al., 1983).

Applying high temperatures on solid particles significantly increases their energetic movement in a random manner (Kirkpatrick et al., 1983). As an improvement method, SA simulates the annealing process and permits random behavior at the beginning of the optimization process to allow investigating solutions in different regions of the solution space. It, in essence, tolerates rather inferior solutions with the objective of finding a better one later during the search process (Pinedo, 2012, p. 385). When the cooling process begins, SA gradually decreases the random behavior of the local search algorithm using a cooling schedule and starts searching in the same region of the solution space instead of jumping to the other one. The start temperature, the cooling schedule, and the number of iterations before changing the temperature are critical parameters of SA. They must be carefully tuned as they have a substantial influence on the quality of the final solution and the efficiency of the search process (Aarts et al., 2005).

The Tabu Search (TS) improvement metaheuristic is another example, which was introduced several years after SA to address the same limitation. From a design perspective, both methods rely on local search methods and have been successfully applied to solve complex combinatorial optimization problems of various complexity, particularly in the area of scheduling Ben-Daya and Al-Fawzan (1998); Adamuthe and Bichkar (2012).

TS was originally proposed by Glover (1989) to enable local search methods to overcome local optima problems. Tabu Search is quite similar to simulated annealing in terms of modification mechanism. It also permits accepting inferior solutions and non-improving steps while inferring a new solution schedule from the previous one (Pinedo, 2012, p. 386).

SA and TS have been widely adopted to address scheduling problems. For instance, Mirsanei et al. (2011) investigated hybrid flow shop scheduling problems with sequence-dependent setup times. The problem was solved considering the minimization of the makespan $\langle HFS_2 \mid s_{j,k} \mid C_{max} \rangle$. In their work, the authors introduced two modification strategies and adopted SA as a control strategy. The first strategy was a classical pairwise job position swap. The second strategy selected two jobs in the sequence and inverted their position to generate the new candidate solution. Both presented strategies were incorporated into the body of their algorithm. The experimental results presented demonstrated slightly improved results compared to the obtained results of a genetic algorithm, which was regarded as a baseline.

A further example that compared the performance of SA and TS for addressing hybrid flow shop scheduling problems was presented by Aurich et al. (2016). The investigated scheduling environment was subject to the family setup time constraint. It was solved to minimize the makespan and the total tardiness $\langle HFS_2(P_4, P_5) \mid f_{g,h} \mid C_{max}, \sum T_j \rangle$. Establishing the complexity of the problem to be NP-Hard was done based on reducing it to the considered identical parallel machine problem by Wittrock (1990).

The core idea of the proposed solution was to decouple the sequencing part of the problem from the allocation part. To address the sequencing in the considered problem, the authors introduced a heuristic method in which the production sequence is constructed. The objective function is to minimize the total number of major setup times under the constraint that the delivery dates of the jobs must be met if possible. To handle the allocation part of the problem, SA, tabu search, and a heuristic were implemented. The authors conducted their evaluation using real problem instances. The SA and Tabu search techniques were based on a single move modification strategy, where a family is selected and reassigned to another machine in the first production stage. Both metaheuristics slightly outperformed the presented heuristic in terms of total tardiness, with the presented heuristic reporting an improved makespan for all problem instances.

Another class of metaheuristics that has been widely applied for solving various combinatorial problems is population-based methods. Genetic Algorithms (GA) is a popular optimization method that is widely adopted to address scheduling problems. The concept of GAs is founded on replicating the biological evolutionary process of species by using a population-based genetic representation and its associated natural phenomena (i.e., mutation, crossover, fitness, and genetic survival) to solve optimization problems. The fundamental elements of genetic algorithms are natural selection and genetic inheritance (Pirlot, 1996). The notion of genetic representation was first proposed by Holland (1975) within the adaptive systems research, later discussed in the context of artificial intelligence, and further extended and applied for solving optimization problems by Goldberg (1989).

Applying genetic algorithms to address scheduling problems depends significantly on the genetic encoding and representation of the problem (Cheng et al., 1996). Each potential solution or chromosome is then evaluated using an objective function to determine its fitness (Ruiz and Maroto, 2006). Following this evaluation, an offspring population of potential solutions must be generated using a selected set of solution individuals. The parents that have been selected are crossed over to reproduce the chromosomes of the next generation, also referred to as 'offspring' (Reeves, 2003). The process of crossover will result in a certain amount of recombination of the parental chromosomes. The offspring solution candidates will iteratively carry the dominant fit characteristics of the parent solutions, mimicking the process of the natural evolution of species (Ruiz and Vázquez-Rodríguez, 2010). To maintain the diversity of solution candidates and to mimic the natural mutation process of species, some of the offspring solution candidates are randomly selected to mutate their genes. The current generation is evaluated before the evolutionary process is repeated. This process is iteratively executed until some breaking condition is met (Whitley, 1994). The premise here is that applying evolutionary concepts may result in the fittest individuals surviving during the emulated evolutionary process. Hence, the best solution individuals maintain the genotypes that express the best characteristics to solve a given scheduling problem.

GA is substantially different from classical metaheuristic methods such as simulated annealing or tabu search, where a population of solution candidates is investigated at each iteration instead of just one solution individual (Geem et al., 2001). Hence, multiple regions in the solution space of a problem can be explored simultaneously. Consequently, this behavior significantly advantages GA or other population-based algorithms over conventional metaheuristic methods. From a solution search perspective, GA can explore more freely in different regions of the solution space and is less likely to be trapped in local optima (Geem et al., 2001). From an efficiency perspective, parallelization technologies can be adopted to accelerate the optimization process using population-based methods (Swan et al., 2022, p. 396).

Genetic algorithms are generally viewed as a more sophisticated approach than conventional improvement strategies, therefore requiring careful encoding of the problem (Cheng et al., 1996; Whitley, 1994). If the encoding process is implemented incorrectly, there is a high chance that the offspring solutions will be infeasible for a given problem (Cheng et al., 1996). Poorly developed encoding can, in essence, lead to a high computational effort to achieve high-quality solutions for a scheduling problem (Andersson et al., 2008; Whitley, 1994). For instance, let us consider a scheduling problem with a single machine with ten jobs. Jobs must be scheduled subject to the sequence-dependent setup time constraint, and the objective is to minimize the makespan. The problem can be encoded as a sequence by modeling the positions of the jobs in the sequence as a chromosome (Pinedo, 2012, p. 389-390). We may apply a single-point crossover that swaps part of the sequence to derive the offspring chromosome from the combination of the two parent genes. The process begins by randomly selecting a piece of of the first parent chromosome. The empty genotypes in the second parent chromosome are then filled in to complete the

offspring chromosome. The same sequence of positions in the second parent is used to fill the missing positions in the offspring solution. The solution becomes infeasible if there is a duplication in any of these positions. We will discuss the role of encoding models in the performance and quality of solutions in the design chapter. In addition, a comprehensive overview of the various crossover operators and the encoding strategies for GA can be found in (Goldberg and Deb, 1991; Whitley, 1994; Picek and Golub, 2010).

For addressing scheduling problems, Reeves (1995) presented one of the earliest works dealing with flow shop problems. The authors investigated minimizing the makespan in permutation flow shop $\langle F_m \mid prmu \mid C_{max} \rangle$. The NP-Hard complexity of the problem justifies the adoption of an evolutionary improvement method such as GA. In the permutation flow shop, the processing sequence in the first stage is maintained and used further in the upcoming processing stages. It implies that the FIFO principles are applied to prioritize jobs starting from the second processing stage. The authors empirically investigated the performance of GA compared to SA and a conventional local search algorithm to solve the problem. As expected, SA and GA proved superior to local search algorithms in solving the problems. Reeves (1995) emphasized in their final remarks the computational efficiency of GA against SA in achieving comparable quality for solving larger scheduling problems (Reeves, 1995, p. 8).

Investigating the same problem, Zheng and Wang (2003) explored the potential of combining GA and heuristic methods to achieve better solutions. The authors similarly examined the minimization of the makespan in a permutation flow shop scheduling environment $\langle F_m \mid prmu \mid C_{max} \rangle$. Unlike the previous approach, Zheng and Wang (2003) relied on NEH constructive heuristic, originally presented by Nawaz et al. (1983), to generate an initial population instead of a randomly generated one. Many authors emphasized the high quality of the solutions obtained by NEH to solve permutation flow shop scheduling problems (Allaoui and Artiba, 2004; Kalczynski and Kamburowski, 2007; Ruiz and Maroto, 2006). Hence, the intention of the authors was obviously to leverage a high-quality starting population to accelerate the optimization process and achieve better scheduling solutions. The encoding of the problem was based on the processing sequence. In addition to conventional evolutionary operators, the authors proposed using SA to control the mutation process. The computational results demonstrated that the proposed method outperformed the NEH algorithm for solving problem instances with up to 75 jobs.

For addressing a rather more complex problem, şerifoğlu and Ulusoy (2004) proposed a genetic algorithm to address scheduling jobs on multiple central processing units. The problem is a hybrid flow shop scheduling problem $\langle HFS_m \mid\mid C_{max} \rangle$. Conventionally, the problem is investigated considering the minimization of the makespan. The NP-hard complexity of the problem can be confirmed by reducing the problem to two-stage and a single machine in the second stage results in the considered problem by Gupta (1988). The genetic representation of the problem by the authors was based on the sequence of jobs in the first processing stage. In essence, GA was utilized to solve the sequencing problem. After the first stage, a simple Last-In-First-Out (LIFO) rule was used to prioritize jobs for processing. The proposed method was evaluated using a benchmark that included up to

100 jobs. The evaluation demonstrated that GA outperforms several priority rules, such as SPT and LPT, in scheduling jobs on the available CPUs.

Oğuz and Ercan (2005) built up on the work of şerifoğlu and Ulusoy (2004) and proposed an improved allocation method to replace the LIFO rule to address job scheduling multiple processors $\langle HFS_m \mid\mid C_{max} \rangle$. While using the same genetic representation of the problem, several allocation heuristics after the first production stage were proposed. They introduced a list-based allocation heuristic that stores a list of the completion time of all jobs in previous stages, which is then used for allocation after the first stage. Oğuz and Ercan (2005) empirically investigated various crossovers and mutation operators and proposed a new crossover operator. The presented computational results demonstrated that GA delivered better scheduling solutions than TS for solving the same problems.

To address scheduling problems in cloud environments, Ye et al. (2019) presented a scheduling method that combines the use of a Genetic Algorithm and due-date-based PDRs for scheduling jobs to a set of virtual machines. The formulated parallel machines scheduling problem is solved to minimize the makespan $\langle Q_m \mid prec \mid C_{max} \rangle$. The authors relied on the EDD rule and assigned intermediate deadlines for the interdependent operations of the jobs to minimize the makespan and reduce the communication time between jobs. The experiments that were conducted included a comparison of the presented method against various PDRs, such as the Min-Min and the Max-Min. The main difference is that the machines' workloads are considered, and dispatched jobs are scheduled to the least loaded machine, which we discussed in both algorithms. The computational results show that the authors achieved superior results for solving small problem instances and lost on performance compared to other rules for solving larger problems.

An example of applying GA to address allocation problems in a virtualized cloud environment was presented by Xu and Fortes (2010). In this paper, the authors used the genetic algorithm to address scheduling virtual machines (VMs) to a set of available physical machines. Unlike many of the discussed contributions, the authors formulated a multi-objective scheduling problem to minimize the overall power consumption, total waste of resources, and further costs. The authors proposed a modified Grouping Genetic Algorithm (GGA) and relied on fuzzy-logic techniques to evaluate different objectives. The proposed GGA algorithm is evaluated in terms of performance, scalability, and robustness against traditional offline bin-packing algorithms through a series of simulation experiments conducted in a wide range of environments. The presented evaluation demonstrated that the proposed method outperforms known bin-packing algorithms and single-objective approaches.

Similarly, Anuradha and Sumathi (2014) demonstrates the effectiveness of adopting GA for allocation problems. In their study, the author conducts a comparative analysis of various resource allocation methods used in cloud computing, as well as their strengths and limitations. The obtained findings demonstrate that GAs can yield better results for resource allocation tasks. GA has also been adopted to address load balancing problems in cloud environments, as presented by Chandrasekaran and Divakarla (2013). The authors utilized GA balance workload in a distributed cloud environment, taking into account

the minimization of rescheduled jobs. The proposed GA calculated the node load before scheduling a job, relying on the fitness function, and provides the best load-balancing solution. The results obtained by the GA show that the performance of the proposed approach is superior to that of the Greedy and Round-Robin algorithms.

Although traditional genetic algorithms can be effective in solving single-objective problems, many real-world decision-making problems require the optimization of multiple business objectives. Scheduling problems are complex environments associated with multiple business objectives that are often of a conflicting nature. Unlike single-objective optimization methods, a multi-objective method does not yield a single best solution. A multi-objective method seeks to find a set of solutions that are better than the remaining solutions in the search space considering all objectives but worse than other solutions in one or several objectives. To address this problem, Srinivas and Deb (1994) proposed a new Nondominated Sorting Genetic Algorithm (NSGA) as a method for solving optimization problems with multiple conflicting objectives. NSGA is based on the principles of genetic algorithms but incorporates non-dominated sorting and sharing mechanisms to maintain diversity and drive convergence toward the Pareto optimal front. The Pareto optimal front refers to the trade-off solutions where no solution can be improved in one objective without compromising at least one other objective. These solutions are also often described as Pareto optimal or non-dominated solutions (Chankong and Haimes, 2008, p. 113).

An important feature of the NSGA algorithm is non-dominated sorting, a sorting method introduced by Goldberg (1989), which works on the principle of grouping solutions into different fronts according to their dominant relationship, where the first front includes non-dominated solutions (Pareto-optimal front), the second front includes solutions dominated only by those in the first front. A solution is considered dominant if it yields a better value for one objective without losing quality in any other objective. This sorting procedure facilitates identifying and preserving the solutions that provide the most optimal trade-offs between conflicting objectives (Goldberg, 1989, p. 198-201). A further essential feature of NSGA is the sharing mechanism, the metric used to calculate the distance between two members of the population, which is embedded to encourage diversity in the population. Hence, it contributes to avoiding premature convergence to suboptimal solutions. This mechanism penalizes solutions that are too near to each other in the search space, thereby prompting the population to explore different regions (Srinivas and Deb, 1994).

Despite the effectiveness of NSGA for multi-objective optimization, it has often been criticized for the high computational complexity of the nondominated sorting, making NSGA computationally expensive, especially when dealing with large population sizes (Srinivas and Deb, 1994). In addition, scholars have pointed to the lack of elitism in the algorithm, which has been known to accelerate the performance of the GA significantly. They also advocated for an automated procedure to mitigate the manual setting of the sharing parameter. To address the limitations of NSGA, its improved variations like NSGA-II Srinivas and Deb (1994) and NSGA-III Deb and Jain (2014) were introduced.

In the presented NSGA-II, the sharing mechanism was replaced with a crowding distance method to preserve the diversity of the solutions. The crowding distance is a metric that quantifies how "crowded" a solution is considering its proximity to its neighbors in the objective space. Solutions with higher crowding distances are favored as they inhabit less dense regions of the objective space, therefore resulting in a more diverse set of solutions. Unlike the sharing approach, the crowding distance does not depend on user-predefined parameters to ensure diversity among population members. In addition, the proposed method is less computationally expensive, which mitigates some of the aforementioned limitations of NSGA Srinivas and Deb (1994).

NSGA III was proposed by Deb and Jain (2014) as an improved version of NSGA II to allow simultaneous optimization of more than four objectives. The basic principle of NSGA-III is analogous to the NSGA-II, with the exception that the diversity preservation in NSGA-III is achieved by the use of the pre-defined reference points instead of the crowding distance approach. Reference points represent the set of points in the objective space that assist the algorithm in exploring different regions along the Pareto optimal front. These reference points operate as markers for the algorithm to navigate towards during the optimization process (Vesikar et al., 2018). A set of reference points can be defined following a systematic approach (Das and Dennis, 1998), or specified by the user.

Based on the reference points, the NSGA III allocates each solution to the closest reference point based on its proximity within the defined objective space. The allocation indicates which region of the Pareto front the solution belongs to. Solutions that are located closer to a specific point are considered to be more optimal in regard to the objectives represented by that point. This rationale preserves diversity in the solutions seeking to optimize conflicting objectives. Furthermore, this approach encourages the exploration of Pareto optimal solutions while simultaneously improving the quality of solutions in explored regions Deb and Jain (2014).

A couple of years after the introduction of the NSGA III, Campos Ciro et al. (2016) empirically investigated its performance compared to its predecessor, the NSGA II, in addressing scheduling problems. The computational results indicated a superior performance of NSGA III compared to NSGA II for solving the problems. The NSGA III is often adopted to address allocation problems in cloud environments, for instance (Kharitonov et al., 2023). In this thesis, we investigate single-stage and multi-stage scheduling problems that must be addressed by considering multiple objective measures. For instance, in Nahhas et al. (2022a), we adopted the NSGA III to address hybrid flow shop scheduling problems that were subject to several operational constraints such as priority groups and family setup times $\langle HFS_m \mid f_{g,h},\ D_{pr},\ M_j \mid C_{max},\ T,\ U,\ MS \rangle$. The considered problems are solved to minimize the makespan, the major setup times, the total tardiness, and the total number of penalties. The computational results demonstrated a complete dominance of the proposed method compared to conventional GA with a weighted-sum approach.

2.3.3 Summary of conventional methods

In the previous section, we explored the adoption of conventional solution methods for solving scheduling problems. The analysis started with a brief overview of scheduling problems and their complexity, which prompted scholars and practitioners to resort to heuristic and improvement methods to address scheduling concerns. The majority of complex scheduling problems are NP-hard. There is an enormous number of heuristic and improvement methods in the literature that are proposed to address different variations of scheduling problems. The presented overview does not cover all proposals but rather highlights the design and adoption of constructive and improvement methods for addressing scheduling problems.

However, the majority of proposed methods in the literature are evaluated to address single-objective scheduling problems to reduce complexity. Constructive heuristic methods are efficient techniques for addressing single-objective scheduling problems but may significantly lose performance when complex constraints or additional objective measures are considered. Therefore, under improvement methods, we discussed local search algorithms and some metaheuristics that are often adopted to address more complex problems. Finally, we closed our analysis by discussing population-based evolutionary methods and their utilization to solve multi-objective scheduling problems. Figure 2.6 summarize the explored methods in the previous section and highlight in dark grey the most relevant methods in the context of this thesis.

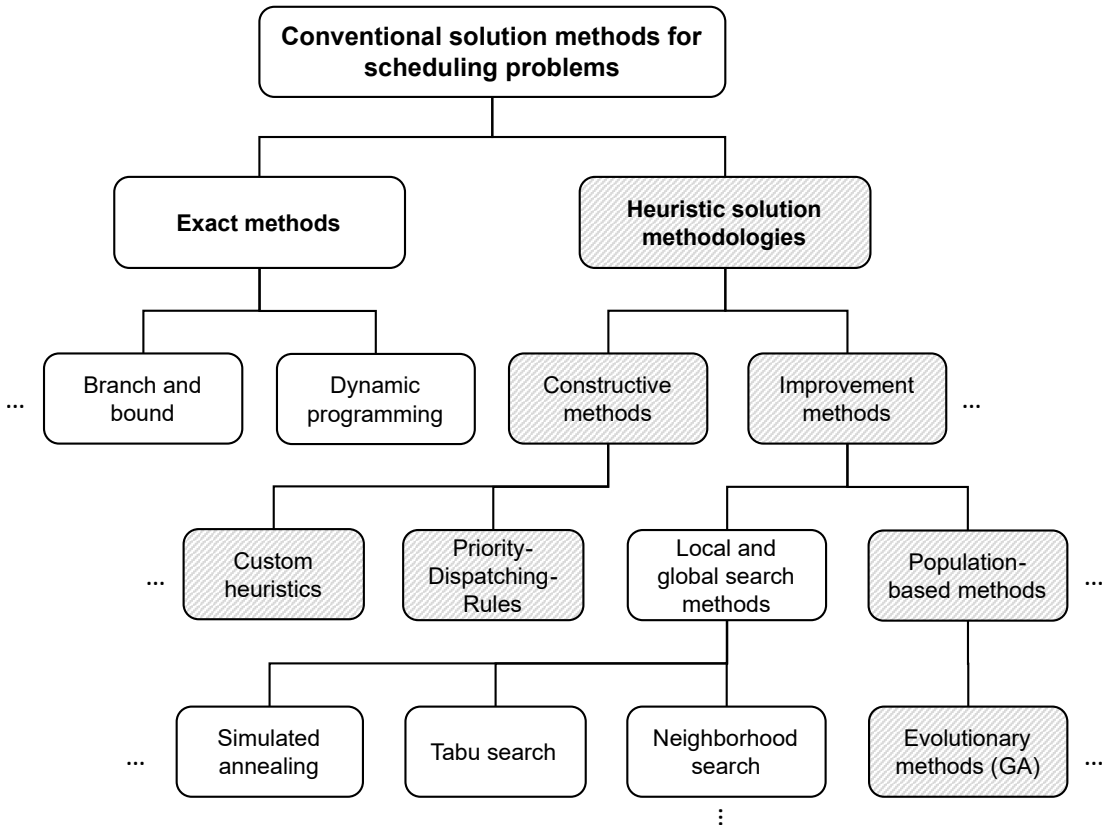


Figure 2.6: Relevant scheduling solution methods for this work.

2.4 Deep reinforcement learning methods

2.4.1 Reinforcement learning

Many hypotheses on learning and intelligence are founded on the notion that humans learn via interaction with their environment. The concept "learning from interaction" lies at the essence of Reinforcement Learning (RL) (Sutton and Barto, 2018b, p. 3-4). Scientific literature offers various definitions of Reinforcement Learning. For instance, Großmann and Poli (1999) views RL as a problem formulation, where the interaction between the agent and the environment is defined by means of states, actions, and rewards. Sutton and Barto (2018b, p. 2) describe RL as a third ML paradigm (along with supervised and unsupervised learning), in which a goal-oriented learning agent interacts with its environment to maximize the reward signal. Szepesvari (2010) characterizes RL as a learning paradigm with the objective of maximizing the numerical performance value.

A typical RL problem is comprised of several elements. As stated by (Sutton and Barto, 2018b, p. 2), two central components of RL are the agent and the environment. The agent is the learner that interacts with the environment and makes decisions (actions) based on the obtained experience (also called observations). The environment is the entity that includes everything outside the agent itself. The agent-environment interaction is divided into episodes or trials. An episode may be seen as a gameplay that terminates when the game is lost or won.

Furthermore, (Sutton and Barto, 2018b, p. 6-7) emphasized such critical RL components as a policy, a value function, a reward, and an environment model. The policy defines an agent's behavior by mapping the observations into actions. The agent's goal is to determine a policy that maximizes the reward. A reward, which is commonly represented by a scalar number, defines the goal of an RL problem and serves as the basis for policy modification. The obtained reward is determined by the agent's action at the moment and the state in the environment. Compared to the reward signal, which delivers an immediate response, a value function calculates the overall amount of rewards an agent can collect, starting from a specific state. Finally, an environment model simulates the behavior of a given environment and enables projections on its potential behavior (Sutton and Barto, 2018b, p. 2).

The fundamental concept of RL can be depicted as Markov Decision Process (MDP). MDP is a mathematical framework for modeling decision-making processes in a certain environment (Buduma and Locascio, 2017, p. 249). It comprises four components that can be represented as a tuple $\langle$ S, A, T, R $\rangle$ (Papadimitriou and Tsitsiklis, 1987b; Kaelbling et al., 1998). The first field of the tuple, S, denotes a set of the environment states (observation space). The second field, A, represents a set of agent's actions (action space), and the third field, T, defines the state transition probability $T\ [s_{t+1} \mid s_t, a_t]$ at a time step $t+1$ given the state s_t and an action a_t the time step t. The last field, R, denotes a reward value $r_t \sim \rho(s_t, a_t)$ obtained by the agent in the state s_t performing action a_t.

2.4.2 Deep reinforcement learning

Despite its earlier success (Zhang and Dietterich, 1995; Tesauro, 1995; Kohl and Stone, 2004), RL methods have demonstrated limited practical achievements in dealing with large-scale, complex real-world problems (Arulkumaran et al., 2017). To address existing constraints, RL methods were frequently combined with different approaches, one of which is Deep Learning. According to (Goodfellow et al., 2016, p. 13), Deep Learning (DL) models are computational representations of biological learning that attempt to mimic how learning may occur in the brain. The part "deep" in deep learning implies that the network consists of multiple layers, resulting in the Deep Neural Network (DNN) (Gustineli, 2022). DL has made a significant improvement in the performance of reinforcement learning for robotics (Levine et al., 2016) and inspired novel applications, such as differentiable neural computers (Graves et al., 2016), asynchronous methods (Mnih et al., 2016), dueling network architectures (Wang et al., 2016) and many others. The success of DL in handling complex tasks and its integration within the existing RL framework resulted in the formation of a new field of Deep Reinforcement Learning. Deep Reinforcement Learning (DRL) is a machine learning technique that integrates RL and DNNs (Cals et al., 2021). The DRL uses DNNs to approximate policy and/or value functions. The approximation becomes crucial when dealing with high-dimensional state spaces, which are common in real-world scenarios. DNNs perform excellently as a function approximation mechanism since they can discover complex structures and patterns, allowing DRL agents to navigate in high-dimensional and continuous state spaces (Yi and Liu, 2023).

A significant milestone in the field of DRL is the presentation of the Deep Q-Network (DQN) algorithm, which was capable of playing various Atari games while achieving human-level performance (Mnih et al., 2013). This work demonstrated the ability of DRL agents to learn optimal policies from the high-dimensional sensory data. The success of DQN was followed by the triumph of AlphaGo (Silver et al., 2016a), in which DNN defeated the professional player in the game of Go. In recent years, DRL has been successfully applied to various problem domains, starting from playing games (Mnih et al., 2013), supply chain problems (Alves and Mateus, 2020), robotics, finance, healthcare, Industry 4.0 (del Real Torres et al., 2022), and scheduling jobs (Hammami et al., 2022; Waubert de Puiseau et al., 2022).

The underlying structure of a DRL problem resembles the one in RL, meaning it can be represented with MDP, with the only difference being that the agent relies on DNNs when defining the policy. The DNN's input layer accepts the encoding of a state as the input data, and the output layer generates an action (or a set of actions) that the DRL agent can execute with regard to the current state. The obtained rewards or penalties for actions are used to tune the parameters of DNNs (Cals et al., 2021). Figure 2.7 depicts an example DRL architecture.

DRL-based (and RL) algorithms can be classified as value-based (off-policy) and policy-based (on-policy) (Alves and Mateus, 2020; Arulkumaran et al., 2017). There are also hybrid, actor-critic-type algorithms that combine elements of both. The main prin-

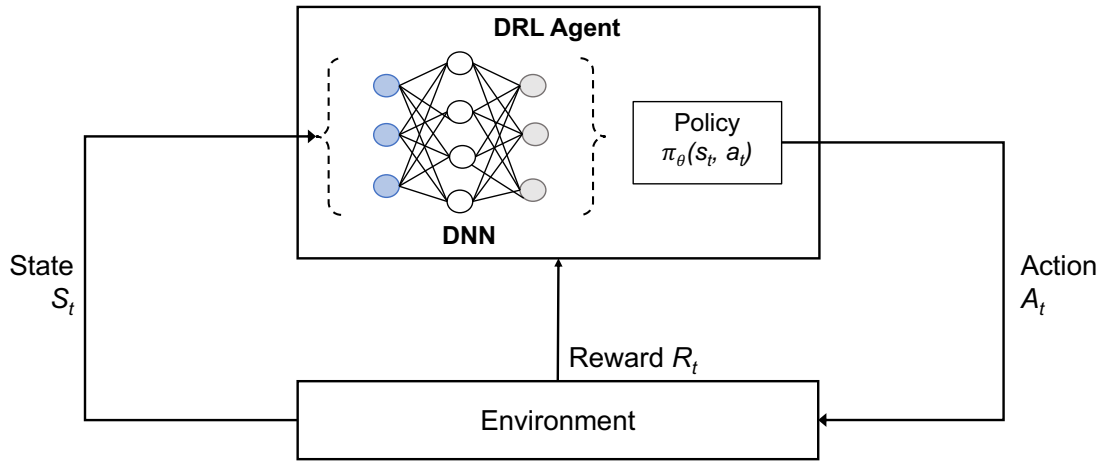


Figure 2.7: General DRL architecture modified based on the RL architecture presented by Sutton and Barto (2018b).

ciple behind the value-based algorithms is to evaluate how valuable it is for the agent to be in the current state (Sutton and Barto, 2018b, p. 58). The evaluation is conducted by computing a value function for each state and action combination. The computed value provides the highest feasible expected return starting from a certain state and facilitates the determination of the policy for the agent (Szepesvári, 2022).

Unlike value-based methods, policy-based algorithms look for the optimal policy directly instead of deriving it from the value function (Alves and Mateus, 2020). The main concept behind policy-based algorithms is to maximize the expected return by continually updating the configurations of the parameterized policy. The update is achieved by applying either gradient-based or gradient-free (also called derivative-free) optimization (Arulkumaran et al., 2017). According to (Schulman et al., 2017a) most policy-based algorithms fall into one of three major categories. The first category includes policy iteration approaches, which alternate between calculating the value function given the existing policy and improving the policy (Bertsekas, 2005; Lee and Sutton, 2021).

The second group includes policy gradient approaches, which employ an estimate of the gradient of the expected return derived from sample trajectories (Peters and Schaal, 2008). The examples of the gradient-based algorithms are REINFORCE (Williams, 2004) and Trust Region Policy Optimization (TRPO) algorithms (Schulman et al., 2017a). Finally, the third category comprises derivative-free optimization techniques such as the Cross Entropy Method (CEM) (Szita and Lörincz, 2006), Covariance Matrix Adaptation (CMA) (Hansen, 2023) and natural evolution strategies (Wierstra et al., 2008), which regard the return as a black box function that can be improved using policy parameters (Szita and Lörincz, 2006). The derivative-free optimization approaches can address a number of practical problems since they produce good results while being simple to understand and apply. However, they tend to scale poorly when the number of parameters increases (Schulman et al., 2017a). Due to the existing limitations of gradient-free techniques, gradient-based learning is still the preferred approach for the majority of DRL algorithms (Arulkumaran et al., 2017).

The third type of DRL algorithm includes the actor-critic method, which combines the elements of both previously described approaches. An actor-critic algorithm learns both policy and a value function, in which the value function is employed to evaluate the policy Wang et al. (2022b). The actor-critic architecture is a frequently used framework that is based on the policy gradient theorem and consists of two interacting components, namely actor and critic Silver et al. (2014). The actor is responsible for determining policy by selecting actions and continually interacting with the environment. The critic, on the other hand, is responsible for evaluating ("criticizing") the agent's actions and is often represented by a state-value function (Sutton and Barto, 2018a, 257-258). The evaluation (critique) is expressed as a Temporal Difference Error (TDE), which is a scalar signal generated by the critic to drive the learning. A positive TDE indicates the tendency towards favoring a particular action in the future, whereas a negative TD error indicates that the tendency should be lowered.

The standard actor-critic architecture includes two main components. The first component is a parameterized approximation function (e.g., neural networks), where the actor learns the policy function. The second component is the critic learner value function, which is eventually used to adjust the actor's policy parameters toward the overall performance enhancement. Combining the policy gradient and the value function methods facilitates leveraging their combined advantages. It somewhat mitigates their individual shortcomings, such as high variation of gradient estimators or poor training convergence (Konda and Tsitsiklis, 2000).

Many advanced DRL algorithms, such as Deep Deterministic Policy Gradient algorithm (DDPG) (Lillicrap et al., 2019), Asynchronous Advantage Actor Critic algorithm (A3C) (Mnih et al., 2016), Proximal Policy Optimization (PPO) (Schulman et al., 2017b), and Twin Delayed Deep Deterministic Policy Gradient algorithm (TD3) (Fujimoto et al., 2018), rely on the actor-critic structure. In this thesis, we focus on PPO and A3C, which we adopted to address scheduling problems. Hence, we will briefly discuss some details of PPO and A3C algorithms.

PPO is a trust-region-based algorithm proposed by (Schulman et al., 2017b) and is considered to be the improvement of the TRPO algorithm. Before looking into the principles of PPO, it is important to explain the TRPO algorithm and the trust region as a concept. TRPO is a policy-based algorithm that optimizes the policy by restricting the update of the new policy within a predefined trust region (Buduma and Locascio, 2017, p. 260). The too-large updates can cause the policy to deviate significantly from the previous one, resulting in training instability and poor performance. To address this problem, TRPO utilizes the Kullback-Leibler (KL)-Divergence to constrain the updates between the old policy and the new policy and prevent large step sizes. The bound on the KL divergence between policies is referred to as the trust region Schulman et al. (2017a). Despite its efficiency and training stability, the TRPO algorithm is prone to a number of drawbacks, such as implementation complexity and lack of scalability. These limitations inspired the development of the PPO algorithm that simultaneously preserves TRPO's strengths while addressing its weaknesses (Schulman et al., 2017b).

Algorithm 5 PPO algorithm, Actor-Critic Style, adopted from (Schulman et al., 2017b)

```
1: procedure PPO
2:     for iteration = 1, 2, ... do
3:         for actor = 1, 2, ..., N do
4:             Run policy π_{θ_old} in environment for T time steps
5:             Compute advantage estimates Â_1, ..., Â_T
6:         end for
7:         Optimize surrogate L wrt. θ, with K epochs and minibatch size M ≤ NT
8:         θ_old ← θ
9:     end for
10: end procedure
```

The PPO algorithm operates by performing multiple training epochs of stochastic gradient ascent on the same data sample to ensure data efficiency. To optimize policy using the PPO algorithm, (Schulman et al., 2017b) propose the clipped surrogate objective function, which computes the probability ratio of actions taken under the old and new policies and clips the ratio within a predefined interval. The proposed clipping mechanism prevents large policy updates and ensures steady convergence. Based on the conducted experiments, the authors conclude that the clipped probability ratio performs better than other variations of the surrogate objective.

Based on Schulman et al. (2017b), Algorithm 5 outlines the procedure of the PPO algorithm with the clipped surrogate objective. Every iteration, each of N (parallel) actors independently interacts with the environment, collects T timesteps of data, and computes the advantage estimates (Algorithm 5, Lines 2-5). The surrogate clip loss function is then constructed using the data collected over T steps and optimized using minibatch stochastic gradient descent for K epochs (Algorithm 5, Lines 7-8) (Schulman et al., 2017b). PPO became one of the most extensively used DRL algorithms due to its straightforward implementation and steady performance, demonstrating its effectiveness in a variety of application fields such as gaming (Schulman et al., 2017b; Cui and Tang, 2023), industrial optimization problems (Cals et al., 2021), autonomous driving (Siboo et al., 2023), and several others.

The A3C is an actor-critic-based algorithm introduced by (Mnih et al., 2016) in 2016 as a lightweight DRL framework that optimizes DNNs by applying asynchronous gradient descent. In contrast to the DQN algorithm, where there is only one agent (represented by the parameterized network) and only one environment, the A3C algorithm comprises a global parameterized network and multiple workers (agents) with its own set of parameters. The workers are being trained concurrently, regularly updating the global network. The parameter updates do not take place simultaneously, which is where the concept of asynchronous originates from. Following every update, the agents return to the global network's parameters and continue the exploration of their environments (Mnih et al., 2016). The procedure of A3C algorithm is demonstrated in the Algorithm 6. Following the initialization phase for a global network and each distinct thread, each worker generates a copy of the global network with the corresponding parameters θ' and θ'_v (Algorithm 6,

Line 1-5).

Algorithm 6 A3C algorithm, adopted from (Mnih et al., 2016)

//Assume global shared vectors θ and θ_v and global shared counter $T = 0$
//Assume thread-specific parameter vectors θ' and θ'_v
1: Initialize thread step counter $t \leftarrow 1$
2: **repeat**
3: Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
4: Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
5: $t_{start} = t$
6: Get state s_t
7: **repeat**
8: Perform a_t according to policy $\pi(a_t|s_t;\theta')$
9: Receive reward r_t and new state s_{t+1}
10: $t \leftarrow t + 1$
11: $T \leftarrow T + 1$
12: **until** terminal s_t **or** $t - t_{start} == t_{max}$
$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{ // Bootstrap from last state} \end{cases}$
13: **for** $i \in \{t-1, ..., t_{start}\}$ **do**
14: $R \leftarrow r_i + \gamma R$
15: Accumulate gradients wrt $\theta' : d\theta \leftarrow d\theta + \nabla_{\theta'} log\pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
16: Accumulate gradients wrt $\theta'_v : d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i;\theta'_v))^2/\partial\theta'_v$
17: **end for**
18: Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
19: **until** $T > T_{max}$

The policy and value function are adjusted after each t_{max} action or when the terminal condition is reached (Algorithm 6, Lines 10-14). In the final stage, the agent calculates the gradients (Algorithm 6, Lines 18-19), after which it asynchronously updates the global shared network (Algorithm 6, Line 21) (Lin et al., 2022). Due to its multiple advantages, the A3C has been successfully applied to various tasks such as gaming and robot control (Sartoretti et al., 2019; Gu et al., 2019), where it has demonstrated great performance compared to other algorithms. Thus, for example, the A3C outperformed DQN in playing various Atari games, additionally demonstrating faster training abilities (Mnih et al., 2016).

2.4.3 Summary of reinforcement and deep reinforcement learning

Deep Reinforcement Learning can be defined as a subfield of Machine Learning that relies on Deep Neural Networks for the approximation of optimal policies and value functions. The formation of DRL as a research field was primarily motivated by the need to overcome

the existing limitations of RL, like handling high-dimensional data and continuous action spaces using the capabilities of Deep Learning. The DRL concept, like its predecessor's, can be defined by means of the Markov Decision Process, with the only difference being that the agent encompasses a DNN that accepts the input data from the state and turns it into the policy. Since DRL is a trial-and-error method, it suffers from the exploration vs. exploitation problem, where the agent is expected to maximize the reward while exploring unknown states. To attain the balance between both, a wide range of algorithms were proposed. The most prominent and widely utilized DRL algorithms are DQN, PPO, and A3C. Due to their relevance to the artifact of the thesis, we dedicated particular attention to the PPO and A3C algorithms by providing an extensive overview of their fundamentals.

The scientific breakthrough of DRL took place when (Mnih et al., 2015) proposed its novel algorithm, known as deep Q-network (DQN), which demonstrated human-level competence in playing Atari games. The DQN's success story was followed by DNN's triumphant victory over a professional Go player presented in the work of (Silver et al., 2016a). The DRL's ability to operate excellently in gaming-like environments motivated researchers to investigate its potential in other domains like robotics (Duan et al., 2016), object recognition (Li et al., 2018), supply chain (Alves and Mateus, 2020), manufacturing (del Real Torres et al., 2022) and many others. Nonetheless, despite DRL's remarkable performance in these domains, examples of its application to optimization and, particularly, scheduling problems remain poorly investigated. In this thesis, we conceptualize, design, and develop DRL scheduling and evaluation models to adopt DRL methods for solving scheduling problems. To identify works related to the presented methodology in this thesis, we conducted a structured literature analysis. The findings of the literature analysis will be discussed in the next section.

2.5 State of the art and literature analysis

In the course of our analysis, we aim to present a systematic comparison of identified solution techniques that exhibit such a combination nature. The analysis is dedicated to providing insights into current trends and future challenges in conceptualizing and designing such solution methods. The objective is to highlight the gaps from a research perspective and suggest solutions from a practical perspective. First, we present a brief overview of the adopted research methodology for conducting this literature analysis. Then, we discuss the search process to highlight our early statistical findings. Afterward, we build on the statistical findings and present an in-depth analysis of selected papers. Finally, we conclude the analysis with a systematic comparison before closing the review with a summary of the main findings.

2.5.1 Research scope and methodology

We argue in our *first hypothesis* the necessity of combining different methods to design adaptive solutions for addressing scheduling problems. Therefore, this section presents

a systematic literature analysis of state-of-the-art scheduling techniques that combines the use of heuristic, metaheuristic, and RL methods for addressing scheduling problems. We reduced our focus to two application fields: cloud and manufacturing environments. In the former application field, the efficiency of such large cloud computational systems heavily depends on the quality of the scheduling mechanisms employed to distribute the workload between available computational resources. The later application field includes well-studied scheduling problems in assembly production systems, such as single-stage parallel machine (PM) or multi-stage Hybrid Flow Shop (HFS) scheduling problems. In such industrial environments, scheduling concerns are inevitable in daily operations. To summarize, the objective of the presented literature analysis is to contribute partially to answering the *first sub-research question* of the thesis at hand.

The adopted methodology to conduct the literature analysis is based on the works presented by (Webster and Watson, 2002) and (vom Brocke et al., 2009). We conducted a Systematic Literature Review (SLR) to answer the *first sub-research question*. The review started with an explorative phase, in which we defined relevant articles and derived search strings. The representation of the results is structured following the recommendations presented by (vom Brocke et al., 2009).

2.5.2 Literature search

Based on the defined scope of the research and presented research question, the conceptualization of the topic is discussed in this section. The goal of the topic conceptualization is to derive appropriate search strings that are then used to query relevant articles from the scientific databases. In order to conduct a structured literature review, search queries, databases, and year ranges are predefined. For analysis, the scientific database Scopus is selected, which references a broad spectrum of peer-reviewed publications and provides access to all major scientific databases and journals (Baas et al., 2020). To constrain the search, the year range between 2010 and 2023 is defined. The selected year range is motivated by the fact that some of the utilized approaches (like DRL-based algorithms) are relatively novel and have been scientifically formed less than a decade ago.

Only peer-reviewed publications in English are subjected to the analysis. Table 2.1 presents the utilized search and refinement search strings in the Scopus database. The original string yielded a large number of publications, 3224 dated back to 21.02.2023. The refinement string is applied to the title, abstract, or keywords to focus the research on the relevant fields of application, namely, cloud and manufacturing environments. Finally, a filtration string is applied to the title, abstract, or keywords of the publications that address multi-objective scheduling problems. Selected articles are subjected to the three filtration stages based on the title relevance, abstract reading, and final examination. We want to explicitly point out that any SLR is subject to several limitations. In essence, it is unfair for unfound contributions to claim conclusive results. For instance, missing the investigation of other scientific databases that Scopus does not index is a limitation of this SLR. Therefore, we conducted statistical analysis on the analyzed application fields

and compared our findings to other reviews such as (Omotehinwa, 2022) and (Pires and Barán, 2015) to increase our confidence in the applied search strings.

Table 2.1: Utilized search and refinement strings.

Description	**Search string**
General search string	Find articles with these terms: (Reinforcement Learning OR DRL) AND (Heuristic OR Meta-heuristic OR Hybrid) AND (Scheduling)
Application field search string	Title, abstract or author-specified keywords: AND (Manufacturing OR Production OR Data-center OR Cloud Computing)
Refinement search string based on the objective of the research	Title, abstract or author-specified keywords (AND (multi-objective OR many objective))

In the first stage, the titles of all initially selected papers are analyzed. If a selected publication is theoretical in nature (e.g., literature review) or its application field falls beyond the scope of our research, it is not deemed relevant. In the second stage, the remaining papers are evaluated based on the relevance of their abstract. An article is not considered relevant if it is clear from the abstract reading that the study has no implementation component and is purely theoretical. Articles that focus on single-objective optimization instead of multi-objective are also excluded. The rest of the publications are subjected to a brief overview in the following screening stage. Publications lacking conceptual clarity or focusing on single-objective optimization instead of multi-objective are excluded from the evaluation. Finally, the remaining publications undergo an in-depth analysis, in which the content of a paper is profoundly examined, and a review is provided.

2.5.3 Statistical overview

Figure 2.8 illustrates a yearly comparison of the initially retrieved publications based on the predefined application field. The horizontal axis depicts the yearly distribution of the discovered publications within the predefined year range from 2010 to 2023. The vertical axis depicts the number of retrieved publications. The orange- and blue-colored bars in the graph represent papers that address different variations of scheduling problems in the cloud environment and manufacturing domains, respectively. The initial observations from the collected results demonstrated a consistent increase over the past five years in the number of contributions in which optimization and reinforcement learning techniques are combined to address scheduling problems. Some breakthroughs in the field of RL can explain these observations, for instance, Silver et al. (2016b). The robust performance of optimization techniques for solving combinatorial optimization problems (Neufeld et al., 2023). Despite the slight predominance of research efforts in the field of manufacturing in

2022, the literature analysis results demonstrated comparable findings in both application fields within the period of investigation.

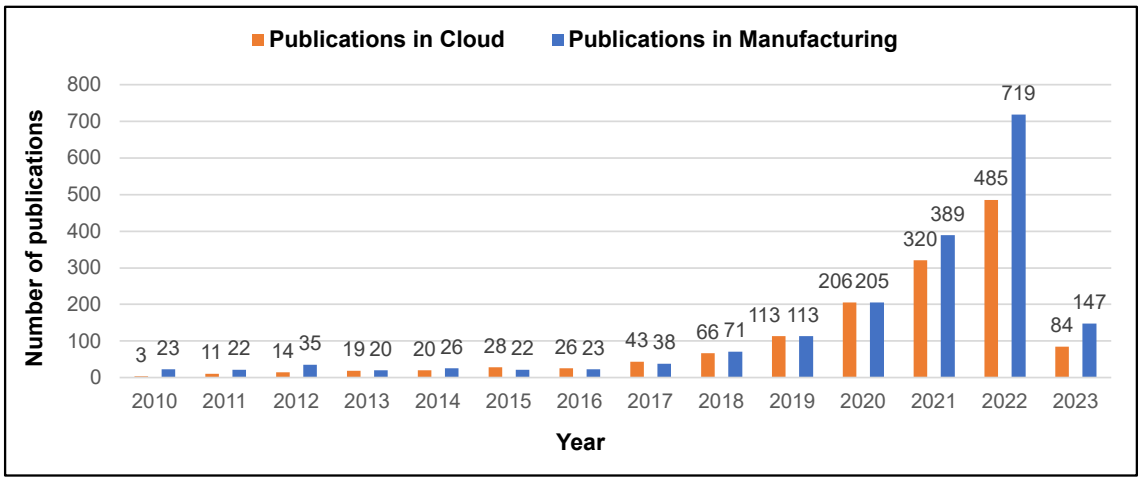


Figure 2.8: Overview of published works in cloud and manufacturing environments.

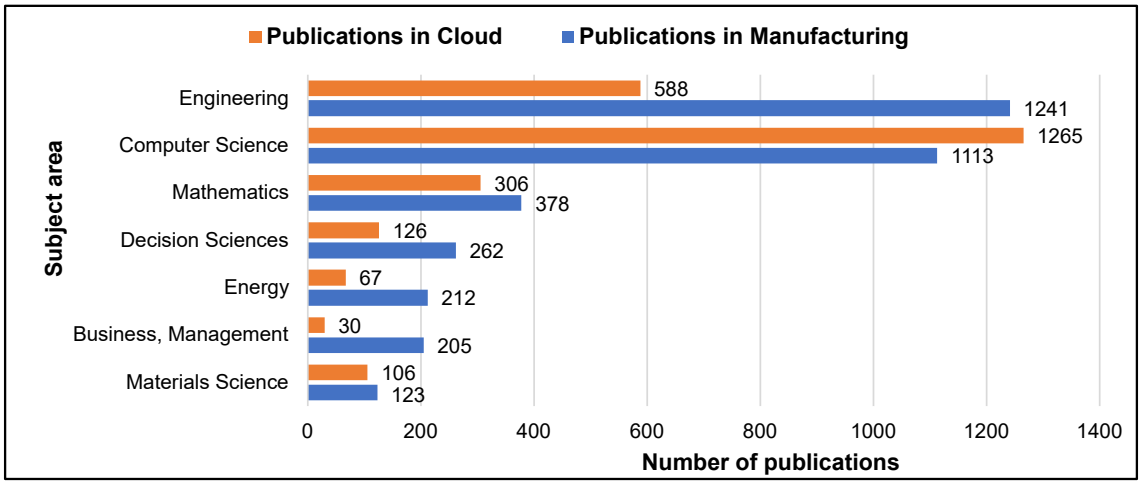


Figure 2.9: Adoption of RL combined with heuristic methods by discipline.

In Figure 2.9, we analyzed subject areas of collected contributions according to the data collected from the Scopus database. The horizontal axis denotes the number of discovered publications. The vertical axis represents the research (subject) areas in which the retrieved publication was published. The orange- and blue-colored bars in the graph represent papers that address different variations of scheduling problems in the cloud environment and manufacturing domains, respectively. The Engineering and Computer Science domains are clearly predominant, followed by Mathematics and Decision Sciences. These statistical results agree with our final findings and contribute to higher confidence in the defined search queries, which we used to retrieve the related works.

The statistical results show that investigating the combination of RL, heuristic, and metaheuristic is more evident for addressing scheduling problems in manufacturing than in cloud environments, as presented in Table 2.2. Scheduling problems in cloud environments are predominantly addressed using heuristic methods. These findings agree largely with

the presented analysis by (Pires and Barán, 2015). The authors stressed the adoption of heuristic and metaheuristic methods, amounting to 81% of identified publications.

Table 2.2: Overview of the conducted breakdown analysis on RL and DRL.

	RL (overall)	DRL	RL-multi-objective	DRL-multi-objective
Cloud	1,436	699	111	51
Manufacturing	1,845	827	163	50
Intersections	57	34	2	1
Total	**3,224**	**1,492**	**272**	**100**

Furthermore, Omotehinwa (2022) presented a bibliometric analysis aiming at investigating the development of scheduling algorithms in cloud and manufacturing environments. While the analysis highlights a robust correlation between scheduling problems and optimization methods like heuristic and metaheuristic, it also reveals a notably weak connection between scheduling problems and the utilization of reinforcement learning or even machine learning, for that matter. We conducted a breakdown analysis to study the adoption of RL independently from DRL methods for addressing scheduling problems. Table 2.2 provides a summary of the analysis. The terms Reinforcement Learning or Deep Reinforcement Learning are often used interchangeably, especially in earlier publications. In this table, the identified publications under DRL constitute a subset of all publications under RL.

We split the process into sequential logical stages to ensure a systematic approach to conducting literature analysis. The process is presented in the Figure 2.10 and Figure 2.11. In the first stage, after defining the overall search string and applying the year range constraint, we extracted the initial list of publications from the Scopus database. The obtained data is further classified into two categories depending on the type of method used, namely RL or DRL. After the completion of the classification stage, the remaining publications are subjected to the third stage, referred to as objective-based refinement. In this stage, we filtered the publications based on the type of scheduling optimization problem and categorized them into single-objective or multi-objective. To maintain consistency, we selected publications with multi-objective problem formulation for each category. The obtained sets of publications were further subjected to the domain-based classification carried out during the last stage of our analysis. Within this stage, the articles were classified into two distinct groups based on the application field. This stage comprises additional internal filtration steps, which can be observed in more detail in the Figure 2.11.

The fifth stage of the analysis was dedicated to screening and in-depth analysis. Furthermore, to examine the relevance of the collected publications, exclusion criteria must be defined. An SLR necessitates clearly defined exclusion criteria to guarantee the reviewer's compliance with the research objective. The exclusion criteria are duplicates, lack of relevance, missing implementation, inaccessible materials, unsuitable application domain, and lack of clarity in the proposed concept.

After all screening steps were completed, we obtained our final set of publications per category. In the coming subsection, we will discuss the final findings and briefly discuss the selected contributions. First, we start discussing selected publications that addressed scheduling problems in a cloud environment. Then, we present a preview of the identified publication that focused on scheduling problems in manufacturing environments.

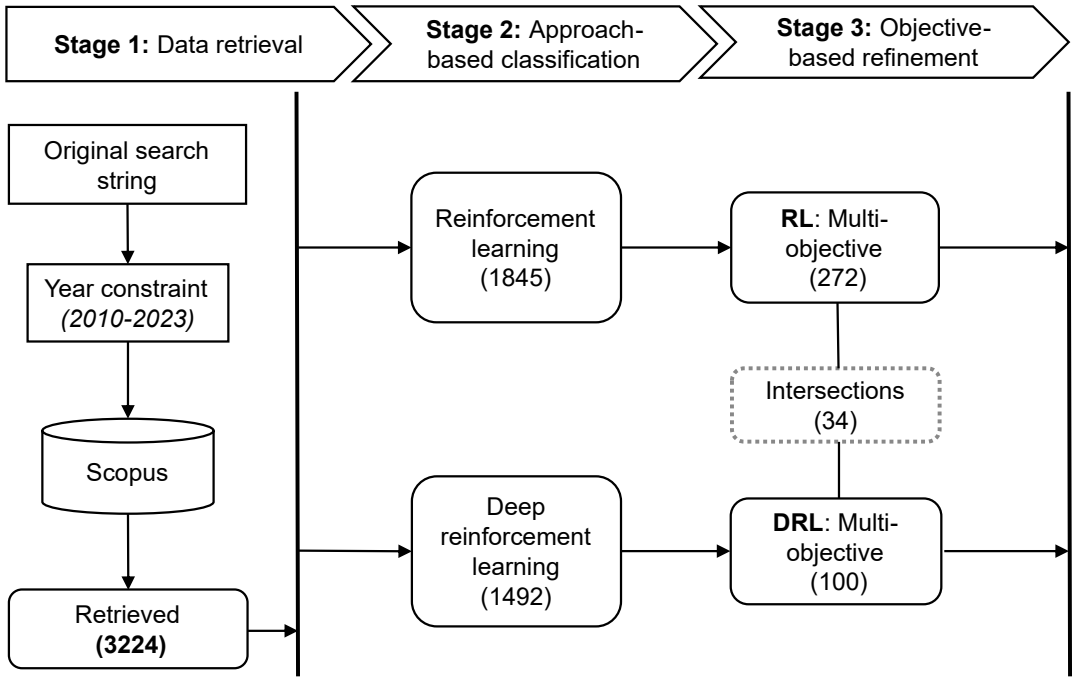


Figure 2.10: SLR publication retrieval, classification, and refinement stages.

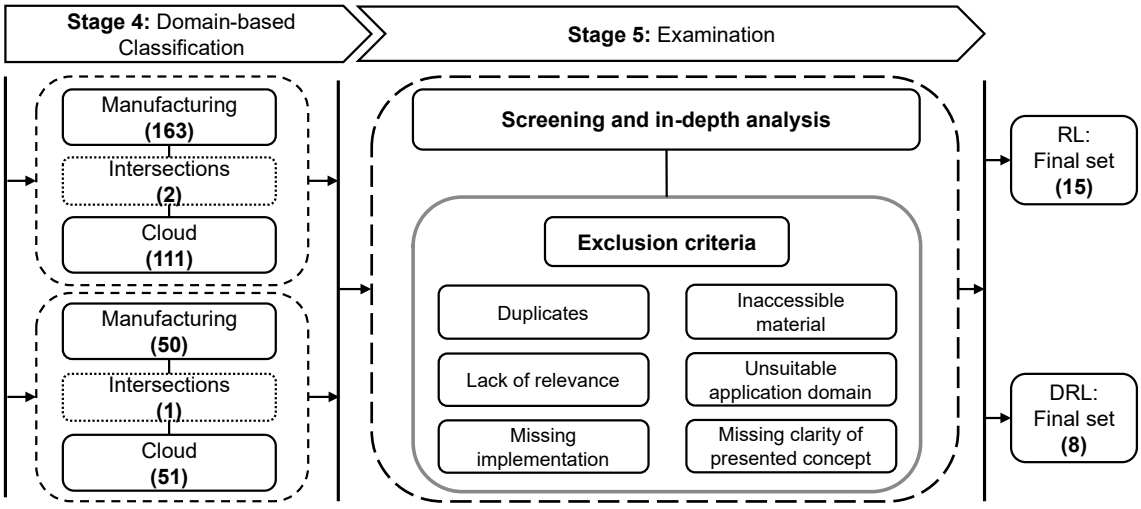


Figure 2.11: SLR domain-based classification and examination stages.

2.5.4 Scheduling problems in cloud environments

The results of the literature analysis reveal many contributions in which conventional ML techniques are adopted in combination with heuristic or metaheuristic techniques for addressing scheduling problems in cloud environments. For instance, Chen et al. (2021) presented a multi-objective resource allocation approach based on the NSGA-II supported by the Autoregressive Integrated Moving Average model (ARIMA) to predict future resource demand. The presented prediction method is based on conventional machine learning techniques relying on historical data to enhance the quality of the virtual machine allocation process and balance the distribution of different workloads between available machines.

A similar but rather more advanced approach is presented in (Jalalian and Sharifi, 2022). The authors proposed a hierarchical multi-objective task scheduling scheme is presented to address parallel machine scheduling problems. The proposed scheme combines the K-means algorithm with the Differential Evolutionary algorithm (DE) for solving the problem considering the minimization of the makespan, which results in optimizing the balancing level. The authors approach the problem by clustering tasks based on their size using the K-means algorithm and load balancing method in order to improve resource efficiency. These clusters are later utilized as the starting population for DE, which is employed to shorten the makespan and balance the load. The results of the simulation suggest the outperformance of the presented approach against two related works for minimizing the makespan. The authors also reported a comparative analysis in terms of workflow overhead and resource utilization. However, our findings indicate rather a marginal presence of RL- and DRL-based approaches in the literature. We found more publications proposing to combine RL or DRL techniques with either heuristic or metaheuristic techniques. Therefore, we discuss these findings in the coming subsections and highlight some of the main learnings.

RL-based techniques

In 2021, an RL-based scheduling approach was presented by Qin et al. (2021) to solve workflow scheduling. The authors considered the minimization of the Execution Costs (EC) and energy consumption while holding to deadline constraints. The presented approach is based on a Q-learning algorithm and employs the Chebyshev scalarization function to optimize the selection of weights. The definition of the action space is discrete, where the agent selects an available virtual machine to execute a task. The authors propose an enhanced version of the Partial Critical Path (PCP) strategy and develop an evaluation metric to assess the quality of generated solutions. The conducted simulation experiments demonstrate that the proposed algorithm outperforms three metaheuristics from related works in terms of execution cost and power consumption.

One year later, Guevara et al. (2022) presented an adoption of RL for solving scheduling problems in cloud environments. The authors proposed an RL scheduling algorithm for scheduling tasks with the objective of minimizing the makespan and overall processing

costs considering Quality of Service (QoS) constraints. The presented RL-based solutions rely on the Q-Learning RL method. The author approaches the problem by comparing the suggested algorithms in terms of scheduling efficiency and makespan, as well as evaluating their performance across seven service classes. The authors use a discrete action space, in which an RL agent decides the number of processing layers where the application is processed. The presented approach is compared against two integer linear programming heuristics, which are adopted from related works. The findings of the research indicate that the RL-based algorithm outperforms heuristics algorithms, especially in intense workload scenarios.

Almost parallel to previous research, Goudarzi et al. (2022) proposed an RL-based approach to solve the resource allocation problem in the data center, taking into account cloud provider and cloud user concerns. The authors relied on a multi-discrete actions space in which an agent allocated virtualized resources to every cloud user. The simulation results suggest that the agent derives appropriate policy to schedule virtualized resources for cloud users, considering the concerns of cloud providers and cloud users. The authors compared the performance of the presented approach against similar techniques in the literature. They confirmed slight outperformance in terms of computational efficiency while maintaining comparable performance in terms of the utility function. However, relying on Q-Learning RL algorithms can be computationally very expensive if the number of users increases.

The findings of the systematic literature analysis show that RL techniques are mainly combined with metaheuristic techniques for designing new approaches. For instance, in (Amer et al., 2022), the authors addressed a multi-objective task scheduling problem in a cloud environment. The authors aimed to minimize the maximum completion time and the execution cost while considering the maximization of resource utilization. The presented approach combines the use of metaheuristic and RL techniques, namely a novel metaheuristic called Harris Hawks Optimizer (HHO) presented originally by Ali Asghar Heidari et al. (2019) and an Opposition-Based Learning (OBL) method that is based on RL in (Tizhoosh, 2006). The presented approach is evaluated using simulation methods and prototypically implemented in the CloudSim simulation package. The conducted evaluation suggests that the presented approach outperforms other metaheuristics, such as Genetic Algorithms or Ant Colony Optimization (ACO).

Similarly, Kruekaew and Kimpan (2022) presented an RL approach to optimize the functionality of a metaheuristic method. The authors presented an Artificial Bee Colony Algorithm (ABC) implementation for job scheduling in cloud environments. To accelerate the optimization and increase the computational efficiency of the algorithm, the authors rely on Q-Learning to enhance the ABC algorithm's computational efficiency. The problem is solved with the objectives of maximizing resource utilization, maximizing the throughput, and minimizing the makespan. The presented approach is compared to some heuristics using a simulation environment. The reported results lack a comparison between a conventional ABC metaheuristic and the presented technique.

DRL-based techniques

In (Gao and Wang, 2022), the author presented a DRL approach for allocating computationally intensive jobs from mobile devices to edge servers. The presented approach is based on DQN with a discrete formulation of actions space, in which an agent is trained to decide whether and where to offload a job to a server. The problem is solved by minimizing the total penalties, which are subject to violations of deadlines. The performance of the presented approach is evaluated in a simulated environment against the baseline algorithm from related works. Experiments show that the presented approach outperforms other algorithms for solving the problem.

In Cheng and Xu (2019), the authors presented a two-phase optimization approach to allocating jobs to computing resources considering the additional computational overhead and energy consumption. The presented solutions combine the use of DQN and PSO as a multi-objective metaheuristic. The first phase uses the DRL agent to find the optimal virtual network mapping schema. After that, PSO is employed to allocate jobs to achieve overall load balancing, energy conservation, and bandwidth minimization. The authors show that the presented approach outperforms some heuristic algorithms such as FIFO, SpeedOut, and Non-SpeedOut for allocating jobs in cloud environments. Although the evaluation is conducted on a server with Spark, no evaluation of the training process of the deployed DRL agent is presented.

Finally, Caviglione et al. (2021) presented a DRL-based virtual machine placement method. They considered the minimization of software and hardware outages, co-location interference, and power consumption to solve the allocation problem. Based on the presented notations in Section 2.1.2, we will refer to violations in this constraint by $min(M_j)$. The author relied on a discrete action space formulation to address the problem. To elaborate, the agent decides at every step which heuristic to use to allocate a new virtual machine. Computational results suggest that the present approach outperforms conventional heuristic methods such as First fit or some variations of the greedy algorithm.

Table 2.3 and Table 2.4 summarize the discussed papers in cloud envrionments. In Table 2.3, we highlighted the considered objective values to solve the considered scheduling problems, the performance, and whether parallelization methods were adopted in designing the proposed methods. In terms of objective values, the makespan, followed by machine utilization and finally by energy consumption, are the most prominent objective measures in addressing scheduling concerns in cloud environments. Except for a single publication, parallelization methods are not considered in the design of modern solution methods.

In Table 2.4, we recapped the design composition in terms of the adopted heuristic, metaheuristic, and machine learning methods of the discussed approaches. The findings of the analysis in cloud environments demonstrated that Q-learning methods under RL are predominant and often adopted to address single-stage scheduling problems. We found no publication that proposed the combined utilization of heuristic, metaheuristic, and machine learning methods. Generally, RL and DRL methods are more integrated with metaheuristic methods to address scheduling problems.

Table 2.3: Summary of literature analysis in cloud environments (I).

Paper	**Objective values**	**Superiority**	**Parallelization**
Chen et al. (2021)	$min(M_m)$, M_{LB}	✓	✓
Jalalian and Sharifi (2022)	C_{max}, M_{LB}	✓	✗
Guevara et al. (2022)	C_{max}, $min(costs)$	✓	✗
Qin et al. (2021)	$min(costs)$, E	✓	✗
Amer et al. (2022)	C_{max}, $M_{utilization}$	✓	✗
Kruekaew and Kimpan (2022)	$M_{utilization}$, C_{max}	✓	✗
Goudarzi et al. (2022)	$max(utility)$	✗	✗
Gao and Wang (2022)	U	✓	✗
Caviglione et al. (2021)	E, $min(M_j)$	✓	✗
Cheng and Xu (2019)	M_{LB}, E	✓	✗

Table 2.4: Summary of literature analysis in cloud environments (II).

Paper	**Solution composition and design**		
	Heuristic	Metaheuristic	Machine Learning
Chen et al. (2021)	–	NSGA-II	ARIMA
Jalalian and Sharifi (2022)	Load Balancing	DE	K-means
Guevara et al. (2022)	–	–	RL:Q-learning
Qin et al. (2021)	PDRs	–	RL:Q-learning
Amer et al. (2022)	–	HHO	RL:OBL
Kruekaew and Kimpan (2022)	–	ABC	RL:Q-Learning
Goudarzi et al. (2022)	–	–	RL:Q-Learning
Gao and Wang (2022)	–	–	DRL:DQN
Caviglione et al. (2021)	Greedy algorithm	–	DRL:Rainbow DQN
Cheng and Xu (2019)	–	PSO	RL:Q-learning

2.5.5 Scheduling problems in manufacturing

RL-based techniques

For instance, Yan and Li (2017) adopted the adaptive heuristic critic method to address small job shop scheduling problems. The presented approach mainly relies on the associate search element and adaptive critic element following the principles of reinforcement learning. The authors presented computational results to compare the performance of the presented approach against related works for minimizing the makespan, maximum tardiness, total tardiness, and number of penalties. The executed simulation experiments show that the suggested algorithm is capable of discovering the best solution through self-adaptation and learning. Two years later, Méndez-Hernández et al. (2019) proposed

a multi-agent RL algorithm to address job shop scheduling problems, taking into account the minimization of the makespan and the total tardiness. The authors rely on discrete action space formulation, in which every agent is responsible for selecting the next operation on its machine. Based on a two-phase approach, the agents are first exposed only to their local environment (single machine). In the second phase, all agents interact to find a compromise solution. The performance of the presented approach is compared to some metaheuristic techniques from related work. Experiments suggest that the X quality of obtained solutions is superior to the compared algorithms.

During the same time, Govindaiah and Pey (2019) presented an RL approach to address challenges of complex production operations like material handling to improve overall efficiency, taking into account multi-objective concerns. Material handling problems are usually formulated as job shop scheduling problems. The formulation of the Q-Learning RL approach is based on discrete actions space, in which an agent makes a change to the material handling plan. The experiments demonstrated that the proposed approach could be very efficient in real-world scenarios and reduce overall costs.

Recently, Zhou et al. (2021) proposed an RL-based scheduler to address dynamic scheduling problems under uncertainty in smart manufacturing. The presented scheduler relies on the Q-Learning algorithm and a discrete action space formulation, in which an agent decides whether to schedule a job or let it wait to solve a single-stage scheduling problem. The authors propose a compositive reward for the presented approach to address multi-objective optimization concerns, taking into account the minimization of the makespan, the minimization of the energy consumption, and the maximization of machine utilization. The conducted analysis demonstrates that the proposed approach increases the system's overall efficiency and somewhat handles unforeseen situations.

To address multi-stage scheduling problems in semiconductor manufacturing, Lee et al. (2019) presented a hybrid technique that combines the use of a QL-based RL approach and standard PDRs. The scheduling problem is solved to minimize the total production, total penalties of the due date, and the number of major setup times. The computational results suggest better performance of the presented techniques compared to commercial solutions.

To address problems of energy efficiency in flow-shop scheduling, a multi-objective model is proposed by Yin et al. (2020). The presented approach is based on reinforcement learning, which is adapted to optimize the update operation of the Grey Wolf Optimization (GWO) Algorithm originally presented by Mirjalili et al. (2014). The authors address the problem by enhancing the suggested algorithm by integrating the features of the Kalman filter with the high efficiency of Reinforcement Learning. The simulation experiments performed on six benchmark problems indicate that the proposed method outperforms the original algorithm for solving flow shop scheduling problems.

Two years later, He et al. (2022) proposed an RL-based approach to address a pure flow shop scheduling problem, which is subject to the minimization of the makespan. Pure flow shop scheduling problems are significantly simpler than the HFS scheduling problems. The authors combine Q-learning with the well-known Nawaz heuristic presented

by Nawaz et al. (1983) to address the problem. The action space of the presented RL learning agent is discrete, in which an agent is trained to select a job for dispatching before relying on the Nawaz heuristic to find the best makespan in the sequence. The presented computational results suggest that the presented approach independently outperforms the pure Nawaz heuristic and Q-learning RL techniques. Another contribution to addressing rather complex manufacturing problems, which involves the selection of manufacturing plants considering various factors, is presented by Chen et al. (2022). The authors integrate different conventional concepts such as domain ontology, the Analytical Hierarchy Process (AHP), and Q-learning-based RL. The framework integrates different spatial and time data of the supply chain network to decide the allocation and scheduling of orders to different production plants. The improved RL technique is used to select the best decision based on the domain ontology and the AHP table to minimize overall costs and maximize efficiency.

DRL-based techniques

In (Leng et al., 2022), a DRL scheduling technique is presented to address a pure flow shop scheduling problem with the objective of minimizing the number of major setup times and total tardiness. The authors rely on a double DQN DRL algorithm to design their solutions, which is evaluated in a case study from the automotive industry. The results suggest that the DRL approach outperforms the NSGA II algorithm for solving the problem. Parallel to previous work, Zhou et al. (2022) proposed a DRL technique to address dynamic job shop scheduling problems in smart manufacturing. The authors introduced a self-adaptive smart schedular that learns to allocate production resources in real-time and enhance decision-making by implementing the DQN algorithm. The authors relied on discrete action space, in which an agent decides to schedule an option or not. The analysis demonstrates that the proposed approach is more efficient at handling real-time jobs and unexpected events than conventional scheduling methods like SPT, FIFO, GA, and RL-based methods. However, the evaluation is conducted on 20 jobs and six machines.

Another example of applying DRL in conjunction with heuristic methods to address scheduling problems is presented by Luo et al. (2022). The authors proposed a hierarchical multi-agent real-time scheduling method that uses the PPO to address the flexible job shop scheduling problem. The investigated problem is solved to minimize the estimated total weighted tardiness and the variance of machine workload in addition to maximizing the average machine utilization rate. The authors approach the problem by implementing three DRL-based agents (an objective agent, a job agent, and a machine agent) to determine temporary objectives, job selection, and machine assignment rules, respectively. The authors relied on a discrete action space formulation. In essence, the agents are trained to select appropriate dispatching rules for scheduling jobs over time. The conducted simulation experiments demonstrate that the proposed method outperforms conventional heuristics such as FIFO, EDD, MRT, SPT, and LPT in identifying achievable objectives

and picking the most suitable rules.

A similar proposal is suggested by Chang et al. (2022), which also combines the use of a hierarchical DRL architecture that controls the selection of different PDRs for solving Flexible job shop scheduling problems. The authors also consider multi-objective concerns by minimizing the total penalties for tardiness and earliness in addition to the total machine load. The architecture of the presented framework comprises two DRL algorithms that are independently responsible for the reward function and the selection of the appropriate PDRs given the system state and objective function. The former DRL agent is designed using Deep Q-Network, while the latter is trained using Dueling Deep Q-Network. The presented experiments indicate that the proposed framework outperforms some heuristics, metaheuristics, and RL techniques in terms of effectiveness and generalization. In (Wang et al., 2022a), a DRL-based scheduling framework is proposed to handle flexible job-shop scheduling problems, which is subject to the minimization of the makespan and overall carbon emission. The presented framework is based on the PPO DRL algorithm. The authors relied on multi-discrete action space formulation, in which an agent selects different dispatching rules that are used for jobs and machine selection. The experiments demonstrate that the proposed model outperforms GA and scheduling PDRs, which are used as a baseline.

Table 2.5 and Table 2.6 summarize the discussed papers in manufacturing envrionments. In Table 2.5, we presented an overview of identified works in terms of considered objective value in addressing the scheduling problem. The majority of problem formation focused on system efficiency business objectives, such as the makespan, machine utilization, and costs. Only two papers considered the minimization of the major setup times and considered family setup or priority constraints. As for customer satisfaction business objectives, six contributions addressed the minimization of the penalties or total tardiness. In terms of design and efficiency, none of the presented works relied on nor reported on adopting parallelization or scalability technologies. Hence, the efficiency and applicability of the methods to address realistic scheduling problems were poorly discussed in the analyzed papers. Unlike identified papers in cloud environments, we cannot establish the superiority of 40 % of identified papers in manufacturing due to a lack of comparisons to related works or other methods.

In Table 2.6, we similarly summarized the design composition of proposed methods in terms of the adopted heuristic, metaheuristic, and machine learning methods. The findings of the analysis are strongly consistent with the identified contribution in cloud environments. Q-learning methods under RL are predominant and often adopted to address scheduling problems in manufacturing. Metaheuristics were poorly adopted in the solution composition, with only two papers. In contrast, four contributions proposed methods combining PDRs with RL or DRL methods. In summary, we did not identify any method that proposed the combined utilization of heuristic and metaheuristic methods for addressing scheduling problems. Roughly 40 % of analyzed contributions proposed adopting pure RL or DRL methods for solving scheduling problems. And finally, only seven publications proposed DRL-based methods for addressing scheduling problems.

Table 2.5: Summary of literature analysis in manufacturing environments (I).

Paper	**Objective function**	**Superiority**	**Parallelization**
Yan and Li (2017)	C_{max}, T_{max}, T, U	✓	✗
Méndez-Hernández et al. (2019)	T, U	✓	✗
Govindaiah and Pey (2019)	$M_{utilization}$, Costs	✗	✗
Zhou et al. (2021)	C_{max}, E	✗	✗
Lee et al. (2019)	$\sum C_j$, U, MS	✓	✗
Yin et al. (2020)	C_{max}, E	✗	✗
He et al. (2022)	C_{max}	✗	✗
Chen et al. (2022)	Cost, $M_{utilization}$	✗	✗
Leng et al. (2022)	MS, T	✓	✗
Zhou et al. (2022)	$\overline{T}$, C_{max}, $\overline{M}_{utilization}$	✓	✗
Luo et al. (2022)	$w * T$, M_{LB}, $\overline{M}_{utilization}$	✓	✗
Chang et al. (2022)	U, $U_{earliness}$	✓	✗
Wang et al. (2022a)	C_{max}, E	✓	✗

Table 2.6: Summary of literature analysis in manufacturing environments (II).

Paper	**Solution composition and design**		
	Heuristic	Metaheuristic	Machine Learning
Yan and Li (2017)	–	–	RL:Adaptive critic method
Méndez-Hernández et al. (2019)	–	–	RL:Q-Learning (multi-agent)
Govindaiah and Pey (2019)	–	–	RL:Q-learning
Zhou et al. (2021)	–	–	RL:Q-learning
Lee et al. (2019)	PDRs	–	RL: Q-Learning
Yin et al. (2020)	–	GWO	RL: Q-Learning
He et al. (2022)	Nawaz	–	RL: Q-Learning
Chen et al. (2022)	–	–	RL: Q-Learning
Leng et al. (2022)		–	DRL: DQN
Zhou et al. (2022)	–	PSO	DRL: DQN
Luo et al. (2022)	PDRs	–	DRL: PPO (Multi-agent)
Chang et al. (2022)	PDRs	–	DRL: DQN (Multi-agent)
Wang et al. (2022a)	PDRs	–	DRL: PPO

2.5.6 Summary of the related works

Based on the findings of the conducted literature analysis, we can observe that in both application fields, the number of publications that address scheduling problems using RL and DRL techniques is rather low, emphasizing a research gap in these research areas. Table 2.3, Table 2.4, Table 2.5, and Table 2.6 demonstrated the overall summary of the conducted literature review. None of the identified publications proposed an integra-

tion heuristic, metaheuristic, and DRL methods. In fact, originally, we did not intend to investigate the adoption of RL methods. However, we found a very limited number of contributions that proposed the DRL method to address multi-objective scheduling problems.

The majority of the authors combine RL/DRL methods with either a heuristic or a metaheuristic. Most of the reviewed works investigated DRL and RL methods from theoretical and conceptual perspectives and reported to known game-like benchmarks for evaluation. It is worth noting that only one contribution discussed the adoption of and integration of parallelization technologies. Scalability and parallelization are essential features in the design of modern scheduling solution methods to support efficient and accurate decision-making processes. Finally, none of the identified works evaluated the presented methods to address real scheduling problems.

2.6 Summary of theoretical foundations and literature review

In this chapter, we presented the required theoretical foundations for designing the research artifact. The section started with an overview of scheduling preliminaries to formulate scheduling problems systematically. The presented methodology of this work combines the utilization of simulation, heuristics, metaheuristic, and deep reinforcement learning methods to address scheduling problems. The objective is to leverage their advantages and avoid their potential disadvantages. Therefore, the second section of this chapter briefly introduced simulation methods that are powerful for modeling complex scheduling environments.

The third section was dedicated to discussing the conventional solution methods for solving scheduling problems with a focus on heuristic and improvement methods. The overview highlighted the adoption of these methods for addressing scheduling concerns in cloud and manufacturing environments. In cloud environments, single-stage scheduling solutions were often presented in the literature. In manufacturing environments, rather complex multi-stage scheduling solutions are required. Based on the established overview of conventional techniques, the preliminaries of deep reinforcement learning were discussed in the fourth section. Since its breakthrough in 2015, marked by the debut of the DQN algorithm (Mnih et al., 2015), DRL has emerged as a promising research field that attempts to address the limitations of its predecessors and offers perspective to deal with business problems exhibiting an optimization nature. DRL methods had been successfully adopted to a wide range of application fields, including object recognition (Li et al., 2018), gaming (Heinrich and Silver, 2016; Mnih et al., 2013), supply chain management (Alves and Mateus, 2020), natural language processing (He et al., 2016; Li et al., 2016), robotics (Duan et al., 2016), healthcare (Lakhan et al., 2023; Dai et al., 2022), or Industry 4.0 (del Real Torres et al., 2022).

Despite its notable achievements in other fields, the adoption of DRL methods for addressing scheduling concerns remains poorly researched. Therefore, the last section presented the findings of the structured literature review on the combination of heuristic,

metaheuristic, and DRL methods for solving scheduling problems. The overall literature analysis demonstrated a limited adoption of these methods for addressing practical and real-world problems. The conducted literature analysis on conventional and advanced scheduling solutions pointed to a gap between research efforts invested in scheduling solutions and their application in the industry.

These findings are consistent with similar concerns that have been raised repeatedly by many scholars over the past 25 years, (Reisman et al., 1997, p. 326), (Ruiz and Vázquez-Rodríguez, 2010, p. 21), (Neufeld et al., 2016, p. 70), (Urquhart et al., 2019, p. 1345), (Romero-Silva et al., 2022, p. 4), or (Swan et al., 2022, p. 400). We also observed that the majority of proposed methods addressed single-objective scheduling problems. In addition, except for a single work presented by Chen et al. (2021), none of the analyzed contributions adopted parallelization and scalability technologies, which inherently limits their adoption for addressing real scheduling concerns (Swan et al., 2022, p. 400). To address some of these challenges, we presented a novel scheduling methodology that combines the use of simulation, heuristic, improvement, and DRL methods for solving scheduling problems. The design of this methodology will be thoroughly discussed in the next chapter.

3 MESEAS: Methodology for Self-Adaptively Solving Multi-Objective Scheduling Problems

This chapter follows a similar flow to the previous one. It comprises six sections, which lay out the design of the artifact presented in this thesis. Based on the identified challenges in the previous chapters, the functional and non-functional requirements of the presented methodology will be discussed in the first section. The detailed requirements and specifications are then summarized and grouped into functionality layers, which the artifact may fulfill. The second section presents the initial blueprint of the methodology that generally describes the rationale and the flow of information between various modules. The third section details the presented abstract representation and presents the reader with the design of the presented methodology using UML component diagrams. Based on the overall design of the presented methodology, every two components of the artifact are detailed in an independent section. Following the same structure as the second chapter, the reader is presented with the modeling, simulation, heuristic, improvement, and machine learning components of the artifact in the remaining sections.

The fourth section details the design of the modeling and simulation components of the presented artifact. The fifth section describes the rationale of the heuristic library component. This section provides an overview of developed and adopted allocation and sequencing heuristics. The sixth section is dedicated to outlining the overall design of the optimization and machine learning components of the presented artifact.

3.1 Design requirements of the artifact

We may define requirements for the intended artifact based on the adopted research methodology presented by von Hevner et al. (2004). A research artifact's main functionalities must be derived carefully from the needs of application environments and supported by scientific foundations from the knowledge base. The conducted analysis of conventional solution methods shows that there is a gap between the results of research efforts in the field of scheduling theory and their industrial adoption (cf. Section 2.3). Furthermore, our findings in Section 2.5 are consistent with several issues that are raised by (Romero-Silva et al., 2022, p. 4; Ross, 2005, pp. 530-531; Reisman et al., 1997; and Maccarthy and Liu, 1993).

Current existing methods lack the flexibility to address the dynamic nature of scheduling problems in various environments. One could also observe oversimplification of business needs, resulting in, for instance, neglecting the consideration of conflicting objective measures, lack of automation to address changes in real environments, or lack of scalability in presented methods. In fact, the notion of the performance of an IT artifact is still debatable since scholars and practitioners have yet to agree on which functional and non-functional requirements are crucial for an accurate definition of performance. For instance, scalability and efficiency requirements are not clearly defined from an IT system perspective. In collaboration with colleagues, different functional and non-functional requirements models of performance in IT systems were systematically analyzed (Alwadi et al., 2018). The objective was to construct a relevant numerical performance model which can be used for optimization. We concluded the analysis with a proposal performance requirements model of IT systems. However, the interdependencies between the identified performance requirements remained an open issue, which was addressed in (Alwadi et al., 2019). In this research, the importance of certain requirements, such as scalability, efficiency, and resource utilization, was highlighted. The presented performance requirements model was evaluated in a survey distributed to IT experts and practitioners. Based on the findings of Chapter 2, we will present the functional and non-function requirements of the intended research artifact. We will refer to a Functional Requirement by (F) and to a Non-Functional requirement by (NF).

3.1.1 Functional requiments

The artifact addresses single-stage and multi-stage scheduling problems (F1). In the analysis presented in Subsection 2.1.1, we discussed that many environments are subject to single-stage or multi-stage scheduling practices. Given some operational constraints, scheduling is sometimes carried out independently on different processing stages, especially in manufacturing environments. In other cloud environments, even a single processing stage may be divided into groups, resulting in a federated environment. Eventually, workload scheduling in every group is handled independently from the other. Therefore, the artifact may address single- and multi-stage scheduling problems.

The artifact addresses scheduling problems with heterogeneous machines (F2). Based on the discussed preliminaries of scheduling problems in Subsection 2.1.1, several variations of machine types are typical in real systems. We may have parallel identical machines Pm or parallel machines with different speeds Qm. Often, these details are overlooked while formulating a scheduling problem, resulting in methods that are not well-suitable for real systems (Romero-Silva et al., 2022, p. 2; Reisman et al., 1997, pp. 325-327). Therefore, the intended artifact must allow modeling machines of a heterogeneous and homogeneous nature.

The artifact supports the consideration of operational constraints in addressing scheduling problems (F3). Due to the combinatorial nature of scheduling problems and their

complexity, operational constraints are often abstracted to simplify a considered problem (Romero-Silva et al., 2022, p. 2). For instance, ignoring constraints, machine differences, and other problem abstraction practices leads to developing methods that are hard to apply in real environments (Reisman et al., 1997, pp. 325-327). Therefore, the designed artifact may allow consideration of several prominent constraints, such as family-dependent constraint $\beta = f_{g,h}$ machine capacity constraint $\beta = M^C$, machine breakdown constraint $\beta = brkdwn$, or machine eligibility $\beta = M_j$.

The artifact addresses scheduling problems, considering objective measures to maximize system efficiency (F4). In Subsection 2.1.3, we extensively discussed various objective measures to evaluate scheduling solutions. The overwhelming majority of scheduling methods are developed considering system efficiency measures such as the makespan $\gamma = C_{max}$ or average flow time $\gamma = \overline{F}$ (Ruiz and Vázquez-Rodríguez, 2010, p. 21, Neufeld et al., 2023, p. 3). They have a significant impact on the performance of an investigated scheduling environment. These findings have been reported by Ruiz and Vázquez-Rodríguez (2010, p. 21) and reiterated again after over ten years by Neufeld et al. (2023, p. 3), Houssein et al. (2021, p. 28), or Murad et al. (2023, p. 170). Therefore, the intended artifact may support considering such objective measures and include other important measures. For instance, minimizing the total number of family setup times substantially improves system efficiency but is usually left out by most studies (Neufeld et al., 2016, pp. 70-71).

The artifact addresses scheduling problems, considering objective measures to maximize customer satisfaction (F5). The popularity of the system efficiency objective measures resulted in overlooking customer satisfaction concerns (Neufeld et al., 2016, p. 61). This phenomenon is more evident in the manufacturing stream of research on scheduling problems where objective values such as total tardiness $\gamma = T$, and penalties $\gamma = U$ are adopted seldom (Neufeld et al., 2016, p. 70; Ruiz and Vázquez-Rodríguez, 2010, p. 21; Ribas et al., 2010, p. 1451). In the cloud stream of research, the minimization of penalties over Service Level Agreements (SLA) is often considered when developing scheduling methods. Therefore, the pursued artifact may support various objective values that are designed to maximize customer satisfaction.

The artifact supports addressing scheduling problems subject to multiple objective measures (F6). Most presented methods for solving scheduling problems in cloud and manufacturing environments consider single-objective function (Neufeld et al., 2016, p. 61; Pires and Barán, 2015, p. 161; Ruiz and Vázquez-Rodríguez, 2010, p. 21). Many of the discussed objective measures in Subsection 2.1.3 are conflicting in nature. If we want to minimize the makespan, we might have significant tardiness over a scheduling period. This conflicting nature obviously increases the complexity of a considered scheduling problem (Neufeld et al., 2023, p. 2; Senthil Kumar and Anandamurugan, 2023, p. 4416; Varasteh and Goudarzi, 2017, p. 9). However, scheduling practices, in reality, are subject to many objective measures. Many scholars have discussed the need to investigate further objective

values and highlighted it to be a research gap as suggested, for instance, by Neufeld et al. (2016, p. 70). Therefore, the artifact may support solving scheduling problems that are subject to multiple optimization measures such as $\gamma = T \wedge C_{max}$.

The artifact's design combines the use of heuristic, metaheuristic, and machine learning methods (F7). Throughout our analysis in Chapter 2, we extensively discussed the overwhelming adoption of constructive and improvement methods for solving scheduling problems. Our findings are consistent with the findings presented by Ghafari et al. (2022, p. 1045), Pires and Barán (2015, p. 164), Ruiz and Vázquez-Rodríguez (2010, p. 21), and Ribas et al. (2010, p. 1452). We highlighted in Subsection 2.3.1 the popularity of heuristic methods for solving single-objective scheduling problems. However, their performance in solving complex problems with multiple objective measures is inherently limited by design. The design of most constructive heuristic methods is based on ranking a set of jobs according to some data indicating their significance. Resorting to improvement methods for solving complex scheduling problems is evident in related works (Ross, 2005). However, improvement methods can be quite expensive in terms of computational efforts to find high-quality solutions (Ross, 2005, pp. 530-531).

It is very difficult to develop an algorithm that is suitable at all times over a scheduling period, even within a single environment. The potential of machine learning techniques, especially Deep Reinforcement Learning (DRL), for addressing scheduling problems remains poorly explored. We investigated their adoption in combination with heuristic and metaheuristic methods and summarized our findings in Section 2.5. We argued in our second hypothesis that combining this technique may yield harnessing their advantages and avoiding their disadvantages. Recently, hybridization in the design of modern solution methods has been stressed by many known scholars in the field (Swan et al., 2022, p. 401; Bhattacharyya, 2018; Dey et al., 2018). Combining several methods that are different from a methodological point of view is called multimethodology (Ferreira, 2013, p. 874). The notion *"hybrid"* is actually a quite old one, which was first discussed by Crowston et al. (1963, p. 83). Heuristic constructive methods allow us to obtain instant solutions for a given problem. Metaheuristic methods, on the other hand, achieve high-quality solutions for solving scheduling problems with multiple objective measures. Finally, adopting DRL methods facilitates training a DRL agent, which can self-adapt to provide high-quality solutions. Therefore, the intended artifact may support the combination of heuristic, metaheuristic, and machine learning methods for solving scheduling problems.

3.1.2 Non-Functional requiments

The design of the artifact is scalable (NF1). The computational efficiency of improvement methods is a well-known issue that many scholars have raised in the field. For instance, Ross (2005, pp. 530-531) discussed the necessity of designing efficient scheduling methods for business-related problems. The author highlighted, among other issues, that most existing improvement methods are computationally too expensive for real application. After

ten years, (Pimminger et al., 2014) suggested parallelization techniques with the introduction of the cloud computing model for improving the computational performance of evolutionary metaheuristic methods for optimization problems (Pimminger et al., 2014, pp. 350-351). Our investigation of related works in Subsection 2.5.4 and Subsection 2.5.5 shows that parallelization techniques are not well-established in developing modern solutions for solving scheduling problems. Our findings are consistent with the parallelization issue raised recently by Swan et al. (2022, p. 401), which looked into bridging the gap between research efforts in metaheuristics and empirical practice Swan et al. (2022, p. 400).

In fact, we found a single method that combines a conventional machine learning method and a metaheuristic method for solving scheduling problems with acceptable computational time using parallelization techniques (Chen et al., 2021). Based on our analysis of several scheduling problems in cloud and manufacturing environments in (Nahhas et al., 2021a,b, 2019a, 2018a), it is crucial to adopt proper parallelization techniques for developing modern scheduling methods. Therefore, the design of the intended artifact may be composable following service-oriented architecture to support parallelization of solution search (Swan et al., 2022, p. 401). Achieving parallelization on optimization and simulation levels facilitates leveraging the elasticity characteristic of modern hardware. For instance, the proposed method might be deployed on an in-house infrastructure but support relying on additional cloud infrastructure if we want to achieve a very high-quality solution instantly.

The design of the artifact is well-maintainable (NF2). Modern solution methods for scheduling problems must be easy to maintain and operate. Here, we want to distinguish between maintainability in terms of the logical design of the solution and maintainability in terms of the software architecture of the method. As for the former, every solution method may contain errors or logical loopholes that are usually detected during operation, given the specificity of every environment. For instance, we detected a special use case while evaluating some heuristics for solving many problems, which required adjusting the algorithm's logic to mitigate the issue. Furthermore, we may want to extend the scheduling data model to support further constraints or objective values to comply with new changes in the operation of our environment. As for the latter, the modular design of a proposed method facilitates the modification, update, or further development of various components independently without significantly impacting the design or the utilization of other components in the method. The modularity issue in designing optimization methods has been recently discussed and identified as a challenge that must be mitigated to achieve reproducible solution methods (Swan et al., 2022, p. 397). Therefore, the design of the proposed method may be easily maintainable.

The artifact offers its functions flexibly (NF3). Flexibility in designing modern solution methods for scheduling problems is crucial to address many of the concerns raised by different scholars. For instance, many authors stressed that the obtained solutions through improvement methods are often not intuitive for practitioners (Branke et al., 2015, pp. 21;

Ross, 2005, pp. 530-531). For instance, Priority Dispatching Rules (PDRs) are more intuitive in terms of functionality and expected solution than an improvement method such as metaheuristic (Song et al., 2023, p. 1600). Similarly, Kim et al. (2023, p. 161) even emphasized the importance of interpreting automatically generated PDRs to increase transparency. A lack of intuitiveness may lead practitioners to mistrust obtained solutions through improvement methods Urquhart et al. (2019, p. 1345). For instance, practitioners in manufacturing environments are unfamiliar with complex improvement methods for solving scheduling problems, which causes mistrust in the solutions they deliver Ross (2005, pp. 530-531) and Romero-Silva et al. (2022, p. 4). Simulation methods are not only significant in designing evaluation environments for complex improvement methods but also provide a tool for users to analyze the obtained solution in detail to strengthen trust. Offering various functionalities, starting with widely used simple rules, helps to establish an accepted baseline for practitioners before applying advanced methods. Therefore, the proposed scheduling method may flexibly offer its functionalities.

The artifact is adaptive (NF4). The previously discussed non-functional requirements somewhat contribute to the adaptivity of the proposed scheduling method. An Adaptive solution method necessitates a flexible design, a well-maintainable architecture, and adaptive logical components that can detect changes in the investigated problems and the quality of obtained solutions. Here, we also want to distinguish between adaptivity in terms of logical design and adaptivity in terms of solution architecture. From a solution architecture perspective, an adaptive solution method requires modular architecture, which allows us to adapt new techniques and extend the components of the artifact without modifying the entire solution architecture. From a logical design perspective, the pursued scheduling method may self-adapt to changes in the investigated problem with minimal human intervention. In a profound analysis of naturally inspired improvement methods, Yang (2021, p. 223) emphasized the importance of developing adaptive and self-evolving improvement methods and urged scholars to focus also on the scalability and efficiency of metaheuristic methods.

For instance, one may rely on the simulation and heuristic library to obtain acceptable solutions for a given scheduling problem. In the case of multi-objective concerns, improvement techniques deliver much higher quality but require considerably more computational effort to achieve them. Improvement methods also require change in the design if the problem significantly changes. However, one may train a DRL method to solve a given problem with minimal computational effort, which may self-adapt to changes in the problem subject to further training. It is well-known that DRL methods are subject to generalization issues. Therefore, to design an adaptive scheduling method, it is necessary to rely on both improvement and DRL methods to solve scheduling problems. After sufficient training, a DRL method can deliver high-quality solutions for solving scheduling problems. Once their performance suffers due to a change in the problem, we may rely on improvement methods until further training is completed and high performance is achieved.

Table 3.1: Overview of functional and non-functional requirements

Label	Requirement summary
	Functional requirements
F1	The artifact addresses single-stage and multi-stage scheduling problems
F2	The artifact addresses scheduling problems with heterogeneous machines
F3	The artifact supports the consideration of operational constraints in addressing scheduling problems
F4	The artifact addresses scheduling problems, considering objective measures to maximize system efficiency
F5	The artifact addresses scheduling problems, considering objective measures to maximize customer satisfaction
F6	The artifact supports addressing scheduling problems subject to multiple objective measures
F7	The artifact's design combines the use of heuristic, metaheuristic, and machine learning methods
	Non-Functional requirements
NF1	The design of the artifact is scalable
NF2	The design of the artifact is well-maintainable
NF3	The artifact offers its functions flexibly
NF4	The artifact is adaptive

In summary, we discussed seven functional and four non-functional requirements that are relevant to developing the proposed artifact based on the design science research framework (von Hevner et al., 2004). Table 3.1 gives an overview of all requirements divided into functional and non-functional.

3.1.3 Functionality layers of MESEAS method

The functional and non-functional requirements of the artifact are mapped into functionality layers. MESEAS methodology can be divided into four distinct layers in terms of functionality. As depicted in Figure 3.1, every layer depends on all underlying layers to achieve their full functionality. The dependencies between layers and their associated components are generally demonstrated by the gradient texture fill of their triangle. The intensity of the gradient fill of a functionality layer increases in relation to the dependencies of other layers on it. For instance, all layers depend on the modeling and simulation layer of the proposed method. The improvement potential that could be achieved by solving a given problem eventually increases by employing more advanced improvement methods. The adaptivity and robustness in solving scheduling problems may also increase in relation to the number of combined techniques in the proposed method.

The first layer: The *"modeling and simulation layers"* rely on the meta-data model for scheduling and the simulation component. After formulating and modeling a scheduling problem, one may navigate what-if scenarios using simulation techniques. The design of relevant components to this functionality layer will be thoroughly discussed in Section 3.4.

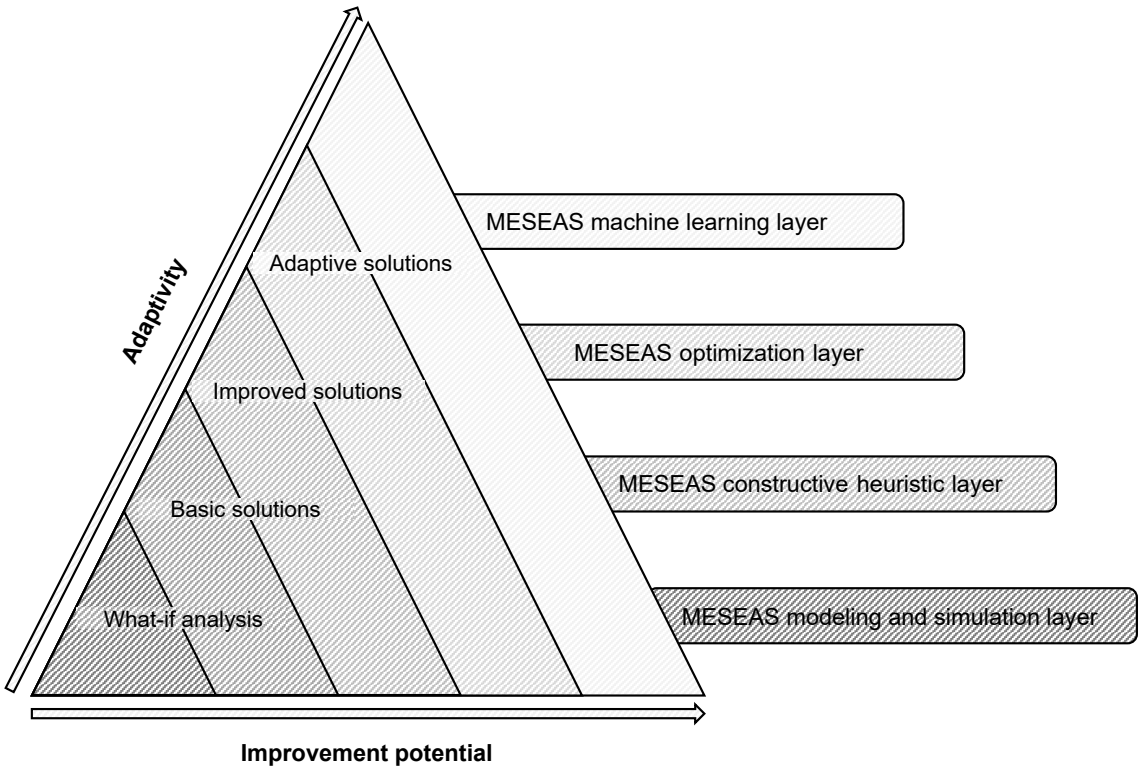


Figure 3.1: Functionality layers of MESEAS methodology.

The second layer: The *"constructive heuristic library"* is loosely coupled with the components in the underlying layer, which allows the investigation of simple solutions for scheduling problems using constructive allocation and sequencing heuristic methods. Both layers partially satisfy the first four functional requirements of the proposed method (cf. *FR1*, *FR2*, *FR3*, *FR4*). Investigating what-if scenarios and receiving intuitive solutions for scheduling problems may establish better transparency and increase the confidence of practitioners in the obtained solutions. It is an issue raised by many scholars in the field (Branke et al., 2015, pp. 21; Ross, 2005, pp. 530-531). These two layers may support investigating scheduling problems with a single objective and sometimes consider an additional auxiliary objective value. The design of components relevant to this functionality layer, along with some allocation and sequencing algorithms, will be presented in Section 3.5.

The third layer: After establishing confidence, one may rely on advanced functionalities to address multi-objective values using the functionalities of the optimization layer. The *"optimization layer"* relies mainly on the optimization component of the proposed method. It may provide access to a set of improvement methods and various encoding models for solving complex scheduling problems. This layer's functionality depends on the method's first and second functionality layers. It may offer modeling and formulation of scheduling problems subject to operational constraints that significantly increase their complexity. These problems can be solved by taking into consideration multiple optimality measures using improvement methods. In conclusion, this layer may fulfill the first four functional requirements fully (cf. *FR1*, *FR2*, *FR3*, *FR4*) and further satisfy the fifth and

sixth functional requirements (cf. *FR5*, *FR6*). The design of this layer and the underlying components will be discussed in detail in Section 3.6.

The fourth layer: The functionality of the *"machine learning layer"* depends in the first place on the machine learning component of the presented method. It also depends on the first and second layers since they are crucial for modeling and evaluation. If we want to achieve adaptive and improved solutions automatically, DRL agents may be trained to learn from solutions to scheduling problems to deliver improved solutions. DRL-based methods may also self-adapt their learned policy if the underlying problem changes with minimal human intervention to deliver improved solutions that are adaptive to changes in the problem. This layer may also depend on the optimization layer if we want to rely on expert experiences for training. We may also rely on the optimization component during further training after carrying out significant changes in the problem formulation, which may lead to a drop in the DRL policy performance. This layer may satisfy the seventh functional requirement (cf. *FR7*) and the fourth non-functional requirements (cf. *NF4*). The design of MESEAS methodology must be modular to satisfy the discussed requirements, especially (cf. *NF1*, *NF2*, *NF3*).

3.2 Conceptual representation of the proposed method

Based on the defined functional and non-functional requirements in the previous section, we may identify various technologies and well-known solution methods to develop a conceptual representation of the intended method. Following DSR, the design of the artifact may rely on the appropriate foundations and applicable knowledge from the knowledge base (von Hevner et al., 2004, p. 80). Based on the literature analysis in *(Section 2.3: An overview of conventional solutions methods)*, the majority of real-world scheduling problems are NP-Hard combinatorial optimization problems. Their complexity evidently motivated scholars and practitioners to favor heuristic solutions to achieve acceptable solutions with reasonable computational efforts. We rely on the heuristic solution methodologies for conceptualizing, designing, and developing several components of the intended method. We further discussed in *(Section 2.5: State of the art and related research artifacts)*, the adoption of Machine Learning (ML) methods, especially DRL methods, to develop self-adaptive solution methods for addressing scheduling problems. Since DRL methods will be utilized heuristically, we extend our overview of conventional methods in Section 2.3 for solving scheduling problems to include DRL under heuristic methodologies as depicted in Figure 3.2. We will revisit our discussion in Chapter 1 to recall a concise overview of the research design of the thesis that investigates three main hypotheses. *The first hypothesis* can be summarized as "A DRL method may learn from solutions of scheduling problems.". *The second hypothesis* can be expressed shortly by "scheduling problems may be addressed by a self-adaptive scheduling method that combines the use of heuristic, metaheuristic, and DRL methods.". Finally, *the third hypothesis* states that "scheduling environments are too complex; therefore, applying multiple heuristic methods over a scheduling period controlled by an improvement method would outperform custom heuristic methods.".

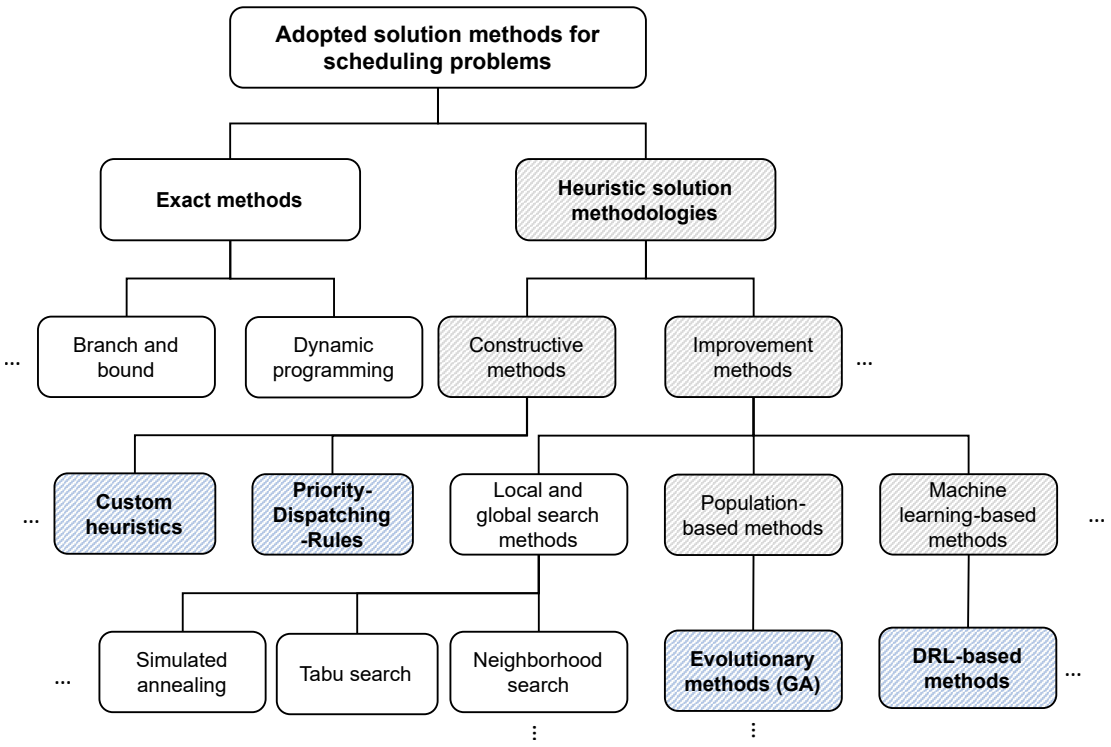


Figure 3.2: Adopted solution methods to develop MESEAS methodology.

To validate the hypothesis of the thesis and comply with the needs of application environments, the conceptual representation of the proposed scheduling methods relies on methodologically diverse solution techniques that are highlighted in blue in Figure 3.2. As depicted in the figure, we focus on heuristic solution methodologies. In the branch of constrictive heuristic methods, we develop and adopt custom heuristics and various PDRs, respectively. As for the branch of improvement methods, we rely on population-based methods, specifically evolutionary ones. Under Machine Learning (ML) methods, we focus our analysis on DRL-based methods since they inherently have some optimization behavior in their design. Validating the previously mentioned hypothesis yields the achievement of the *overall objective* of this thesis and answers the *main research question* (cf. Section 1.3). The thesis's research objective incorporates most functional and non-functional requirements. To comply with the discussed requirements, we follow a component-based approach (Turowski, 2003) in designing the proposed scheduling methodology.

Figure 3.3 draw an abstract representation and an information flow diagram of several key components of the intended artifact. The conceptual representation of MESEAS methodology comprises seven main logical elements, four of which are core-functional components. The core functional components of the proposed methods are *the simulation component, the heuristic library component, the optimization component, and the machine learning component.* Following a bottom-up perspective, we will start with the meta-data model for the scheduling problem. It is required to model and formulate various scheduling problems relying on the discussed preliminaries and objective values. Based on the investigated environment and modeled problem, *the simulation component* may con-

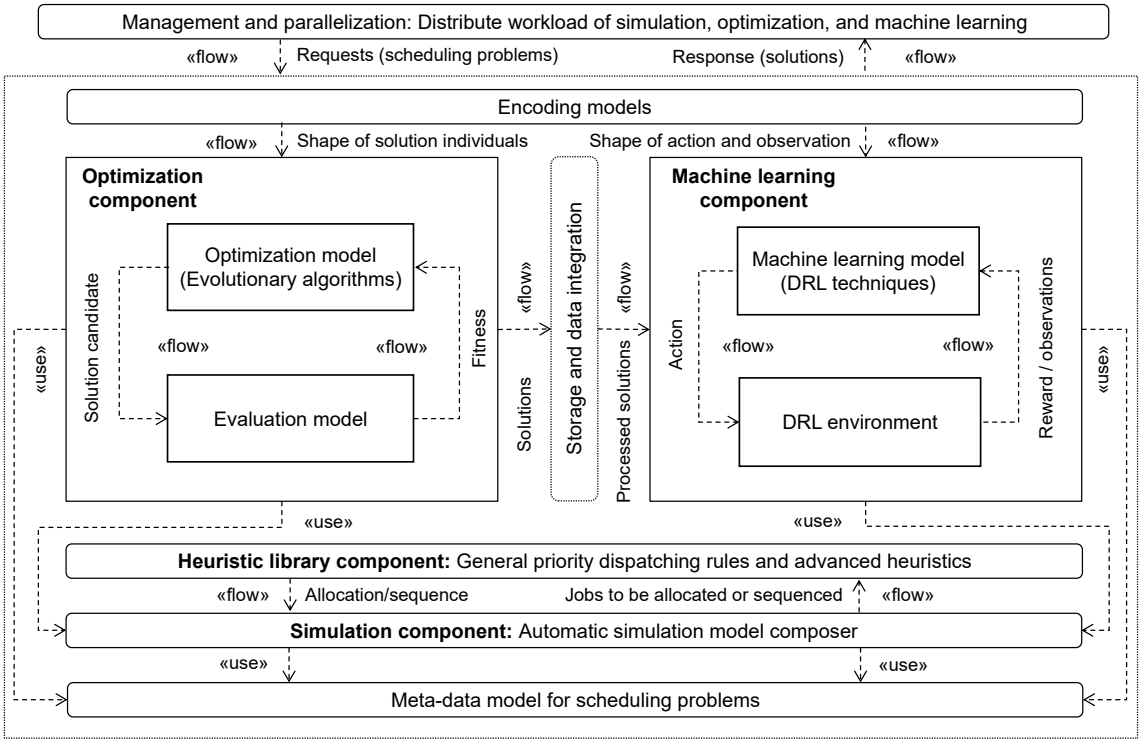


Figure 3.3: Abstract and information-flow representation of MESEAS method.

sume meta-data to build a digital representation of the considered system automatically. We relied on discrete event simulation techniques to design and develop the simulation component. Here, automation is crucial to minimize the modeling effort. Therefore, an automated simulation model composer is necessary to develop the proposed method. In Figure 3.3, we annotate the relationship between the simulation component of MESEAS methodology and the meta-model for scheduling with *≪use≫*. *The heuristic library component* is loosely coupled with the simulation component and may provide access to a set of allocation and a set of sequencing constructive heuristics. The simulation component relies on the heuristic library component to either allocate jobs to machines or identify the sequence by which jobs may be processed on machines.

The third core functional component of MESEAS methodology is *the optimization component*. It may comprise an optimization model and an evaluation model. As we discussed earlier, we intend to rely on evolutionary algorithms to design and develop the optimization model. An evolutionary algorithm such as Genetic Algorithms (GA) depends on an encoding model, which dictates how a GA is used to solve the problem. For instance, a GA may be used to allocate families of jobs to a set of available machines with the objective of minimizing the makespan. Hence, the shape of the genome constituting a solution individual would be the vector that dictates where every family of jobs must be processed. Therefore, the *optimization component* may rely on various encoding models, which are provided by the encoding models component of the intended methodology. Every evolutionary algorithm requires a proper evaluation model to evaluate the quality of generated solutions individuals subject to some objective function. Therefore, the evaluation com-

ponent also relies on the encoding models and the meta-model for scheduling problems to set up a proper evaluation environment for the *simulation component*. The evaluation environment initializes simulation instances to evaluate the quality of suggested solutions by the optimization model. As depicted in Figure 3.3, we annotate the optimization component relationship with the *simulation component* and meta-data model by ≪*use*≫. Upon request, the optimization process starts after initializing the optimization and evaluation models using the meta-data model for scheduling and required encoding. Usually, the first generation of solution candidates is randomly generated and sent to the evaluation model. Given some objective functions, the evaluation model investigates the fitness of the solution candidates and sends them back to the optimization model. This process is repeated until some breaking criteria are met.

Finally, the *machine learning component* of the Meseas Methodology is the fourth core-functional component. Similarly, it may consist of a machine learning model based on DRL methods and a DRL environment. This component may also depend on the meta-data model for scheduling problems, the encoding models, and the *simulation component* to function properly. The meta-data model defines the shape of a DRL agent's observation space and reward function. Meanwhile, the encoding models characterize how the agent interacts with the environment through the action space. Traning a DRL agent is an iterative process that starts after defining the action space, observation space, and reward function. The designated DRL agent takes action, which is processed by its environments and passed further to the simulation component. Based on the action taken, the DRL agent receives a reward and observations of the environment. The reward is usually based on some objective values for solving the scheduling problem. This process is repeated until some breaking criteria are met. As depicted in Figure 3.3, we annotate the relationship of the *machine learning component* with the meta-data model and the *simulation component* with ≪*use*≫. We may also rely on high-quality solutions obtained by the *optimization component* to pre-train the agent. In this case, we need to preprocess the obtained solutions and format them to correspond to the shape of the defined actions and observation spaces before starting to train the agent. In essence, the agent is trained using experiences to imitate how the optimization component would solve the problem.

The proposed scheduling method supports the parallelization of simulation, optimization, and machine learning workload. For instance, GA optimization starts with a population of solution candidates forming a generation. The evaluation of solution candidates can be distributed to multiple simulation instances to accelerate the optimization process. We can also start solving multiple problems, which requires the parallelization of the optimization and machine learning components to accelerate decision-making processes. The overall abstract representation of MESEAS method depicts a general concept to integrate the utilization of heuristic, metaheuristic, and DRL methods for solving various types of scheduling problems. These problems are subject to diverse operational constraints and multiple objective values. The conceptual representation also highlights modularity in the design of the proposed method to increase adaptability and efficiency.

3.3 Component-based design of MESEAS method

Based on the functionality layers and the abstract representation of the proposed methodology, we rely on Unified Modeling Language (UML) to formalize the design of the artifact. UML provides a set of well-established practices to model various aspects of software systems (Seidl et al., 2015, p. 1). It is a recognized molding language that is published by the International Organization for Standardization in the (ISO/IEC 19505-2: 2012, Information technology, 2012). We utilize UML component diagrams (ISO/IEC 19505-2: 2012, Information technology, 2012, pp. 155-164) to represent the overall architecture of the MESEAS method and denominate the design of the core-functional components of the method. Figure 3.4 depicts the overall architecture of MESEAS methodology using UML component diagrams. The core-functional components of the presented methodology may be grouped into three subsystems, namely, *MESEAS Data Management, MESEAS Simulation and Heuristic, and MESEAS Optimization and Machine Learning.* The overall design includes some auxiliary management and user interface components such as a front-end, MESEAS layers manager, and parallelization components. The front-end component provides access to the main entry point of the methodology and connects directly to the Data management Subsystem and experiment controller component. Given a request, the required data is extracted from the database and passed to the *Layers Manager* to schedule a request by the Simulation and Heuristic Subsystem or the Optimization and Machine Learning Subsystem. Parallelization techniques are integrated to execute multiple instances of the simulation, optimization, or machine learning components to accelerate solving a scheduling problem.

MESEAS Data Management Subsystem: As presented in Figure 3.4, this subsystem encompasses two main components: a database and a logging system. These components are responsible for managing data of scheduling problems and results from various other components. The database is used to ensure the consistency of requests and the associated results from various components such as machine learning, optimization, or simulation. The logging system is dedicated to routing data between various components and the front end, which allows monitoring the progress of certain requests before receiving the final results.

MESEAS Simulation and Heuristic Subsystem: This subsystem includes the logical modules of the simulation and heuristic library components as depicted in Figure 3.4. All solution variations of scheduling problems in the MESEAS method are discrete event simulation-based approaches. Therefore, the components of this subsystem are fundamental for the functionalities of all other components. The previous section elaborated in a general sense on how these components interact with each other. Upon request, the *Simulation Environment Manager* processes configuration and passes the required met-data for a scheduling problem to the simulation component, which builds a simulation model of a given environment. Based on the intended analysis, it is integrated with the *Heuristic Library* component to facilitate investigating constructive solutions for a scheduling problem. The heuristic library component offers access to two sets of alloca-

tion and sequencing algorithms. We developed some allocation and sequencing algorithms for scheduling problems and adopted many other heuristics from the literature.

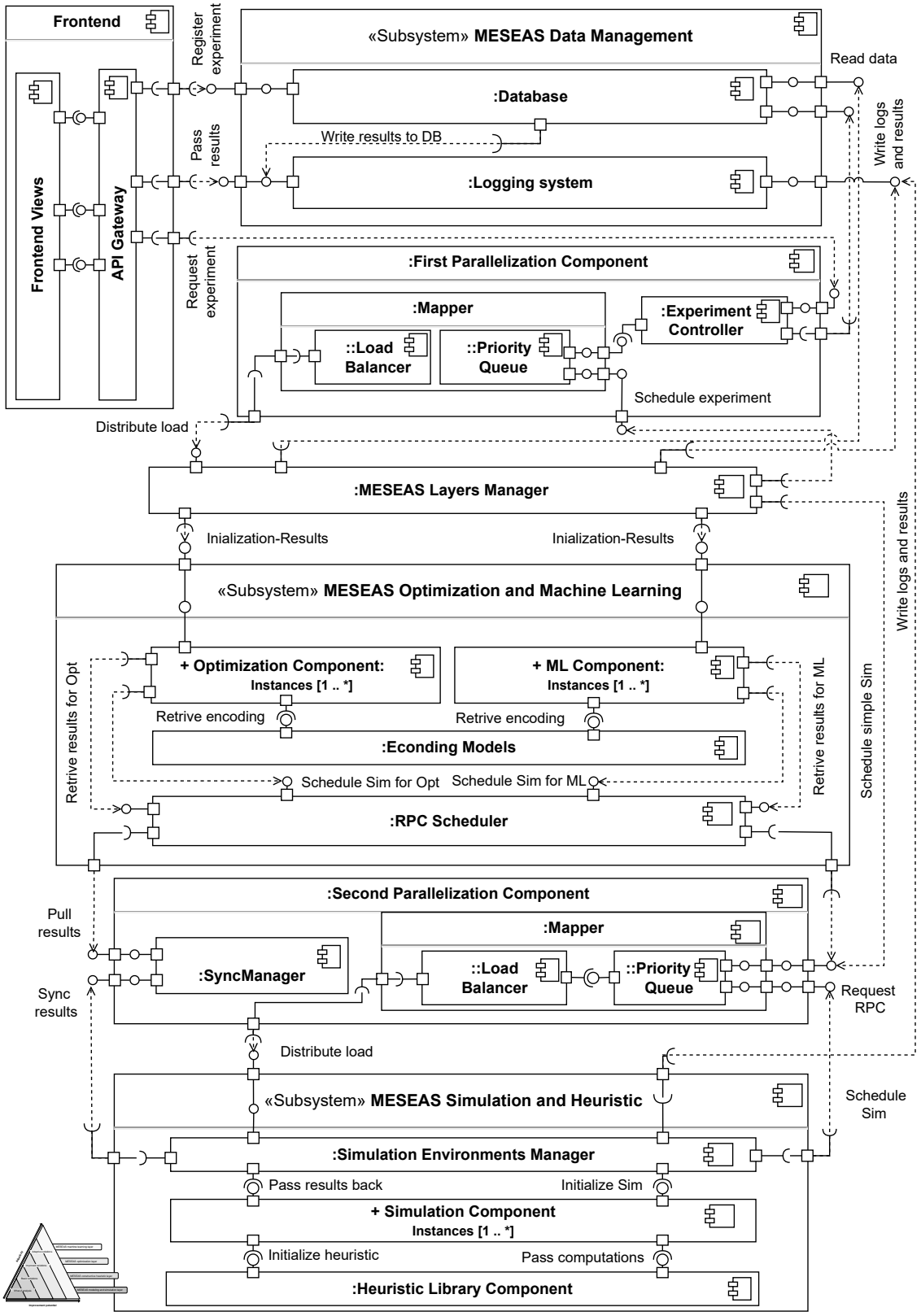


Figure 3.4: High-level UML component diagram of MESEAS methodology.

MESEAS Optimization and Machine Learning Subsystem: As shown in Figure 3.4, this subsystem contains the optimization and DRL logical modules. Both components rely on the encoding models component of the method. Three main encoding models were developed and adopted in the design of the proposed method. An encoding model defines how an optimization or a DRL algorithm is adopted to solve a scheduling problem. Every encoding model allows the investigation of a scheduling problem from a distinct perspective. For instance, GA can be utilized to allocate families of jobs to machines and then rely on some sequencing algorithm. We could also adopt a DRL encoding model to select various allocation and sequencing algorithms over a scheduling period to solve a scheduling problem. Such encoding would yield a hybrid approach, which combines the utilization of constructive and DRL methods for solving the problem. Generated solutions or actions taken from the optimization and ML components, respectively, are passed to the Remote Procedure Call (RPC) scheduler. This scheduler passes generated solutions to the Simulation and Heuristic subsystem to be evaluated using available simulation instances.

Figure 3.5 presents a UML sequence diagram for the overall architecture of the presented method. The figure is annotated with the functionality layers of the presented artifact. It depicts how the simulation and heuristic subsystem, as well as the optimization and machine learning subsystem, are triggered. The front end contains several views for configuring requests to solve a scheduling problem and monitoring the progress of optimization or machine learning components in searching for solutions. As shown in Figure 3.5, the request is received through the experiment view of the front end. The front end allows the user to model a scheduling problem based on the meta-data model for the scheduling problem of the presented method. Based on the problem, the experiment interacts with the database to provide access to a set of configurations for the available functionalities provided by the various layers of the proposed methodology. Upon completing the configuration of the problem and desired functionality, the request is scheduled for execution through the messaging system of the experiment controller. The request is forwarded further to the layers manager, which initializes various components and routes the request to the appropriate subsystem. The proposed methodology fulfills requests of its underlying functionality layers using two core subsystems, namely, *≪MESEAS Simulation and Heuristic Subsystem≫ and ≪MESEAS Optimization and Machine Learning Subsystem≫*.

The core functional component of each subsystem of the presented method will be discussed in the coming three subsections. We align the objective of the thesis and discuss requirements with the design of the thesis artifact. Based on the component diagram of MESEAS methodology, we will discuss in detail the design of the modeling and simulation components in Section 3.4. Following, Section 3.5 discusses the most important allocation and sequencing algorithms that are developed during the design of the methodology. Finally, Section 3.6, presents an overview of the developed optimization and machine learning components of the proposed methodology.

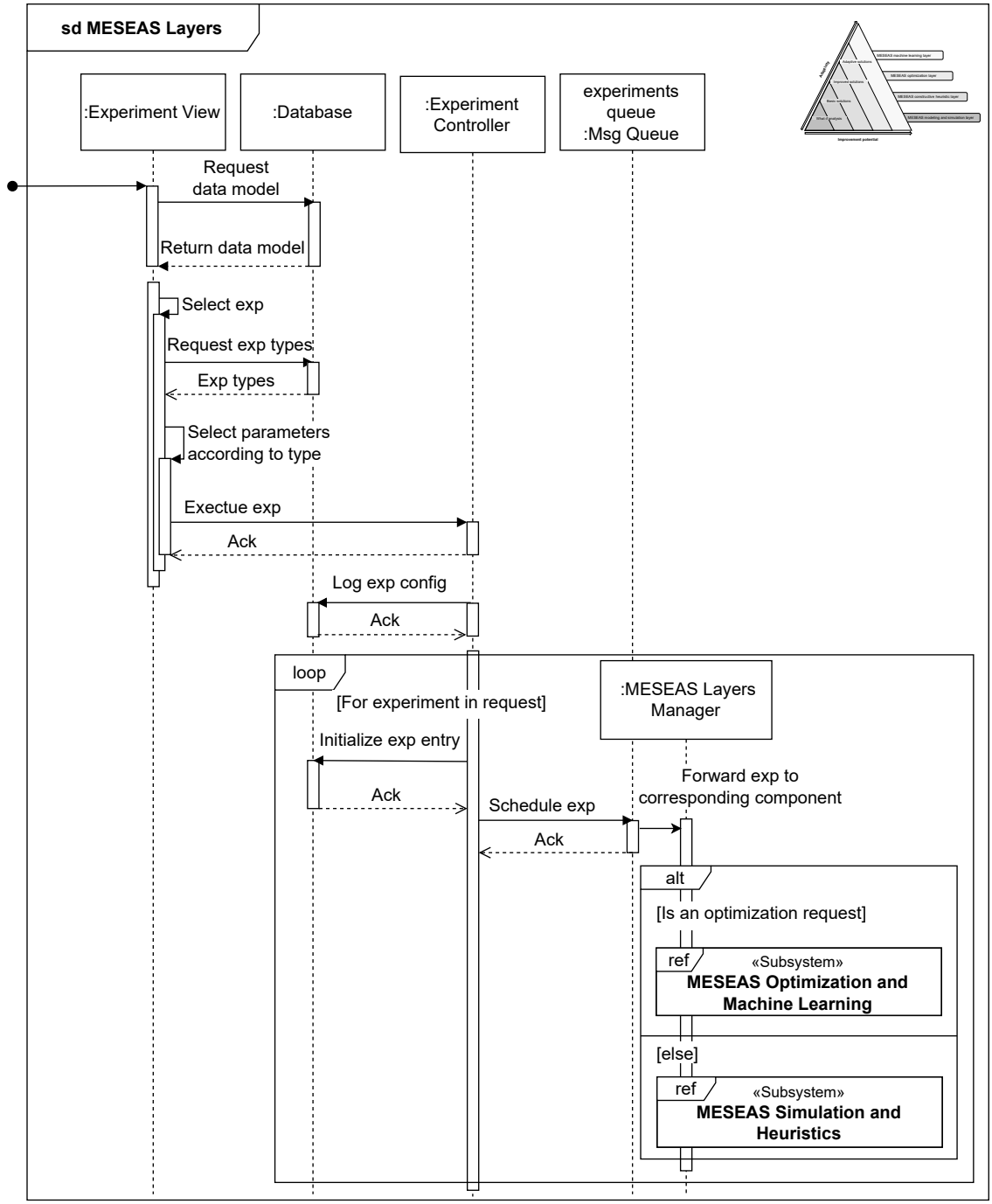


Figure 3.5: UML sequence diagram of the presented methodology.

3.4 Modeling and simulation components

3.4.1 Meta-data model of MESEAS method for scheduling problems

The developed method relies on a data model, which is developed based on the presented preliminaries of scheduling problems in Section 2.1. Figure 3.6 summarizes these elements with some relevant examples for formulating single-stage or multi-stage scheduling problems. The data model describes the physical structure of a considered environment and can be divided into structural and behavioral data. The upper part of Figure 3.6 depicts the structural part of the data model, which expresses the machine environment and the relevant operational constraints. For instance, given that our considered physical environment contains two processing stages, multiple machines are available in parallel at each stage. The lower part of the figure describes how the modeled physical system should behave during a scheduling period. For example, given that we have a set of jobs, we want to process them to minimize the makespan and total tardiness. The MESEAS method incorporates multiple variations of processing stages, machine type, and operational constraints, facilitating the modeling of a considered scheduling environment.

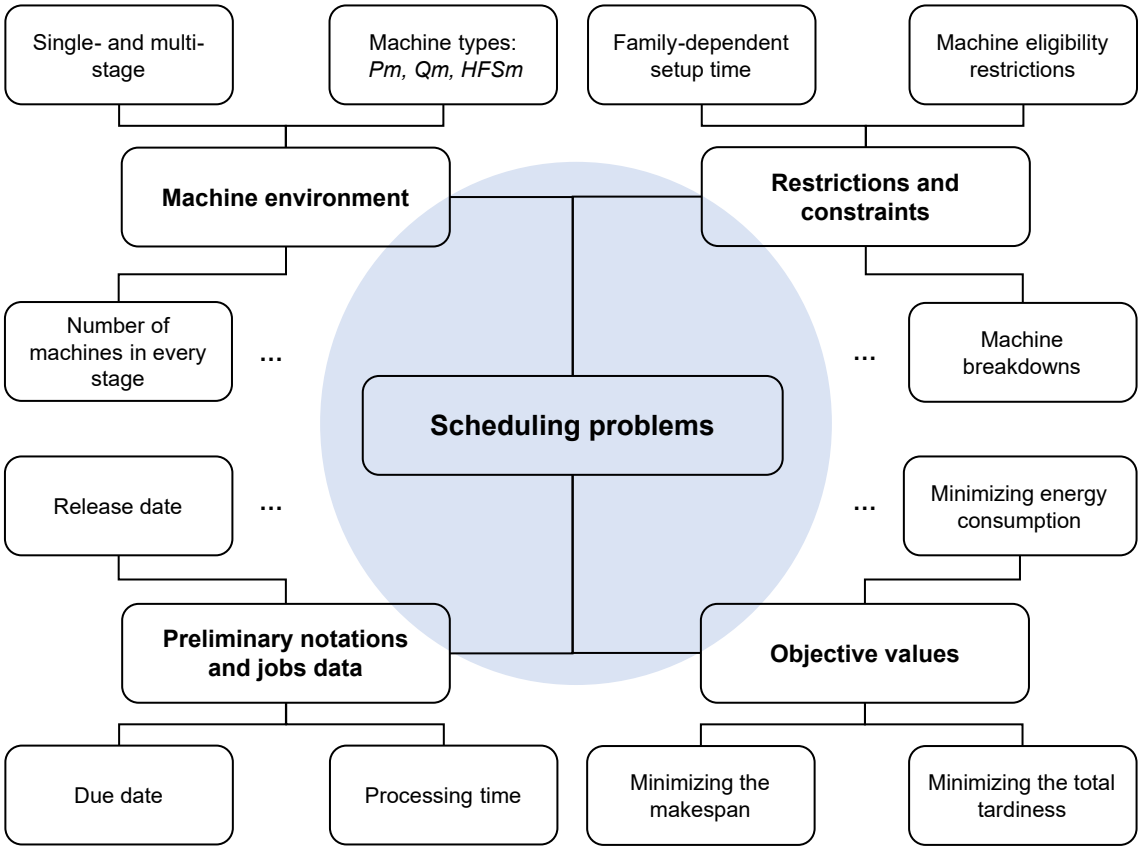


Figure 3.6: Meta-data model of MESEAS method.

3.4.2 Data model of MESEAS method for scheduling problems

The presented data model relies on the triple notation $\langle \alpha \mid \beta \mid \boldsymbol{\gamma} \rangle$ presented by Graham et al. (1979). The data model comprises data structure, machine environment, possible constraints, and objective functions. In the coming subsections, we will discuss every element of the data model in great detail. Based on Graham et al. (1979) notation, we may define a Scheduling Problem (SP) such that $SP = \langle \alpha \mid \beta \mid \boldsymbol{\gamma} \rangle$. Given some structural data that describe the machine environment α, the data structure is required to formalize a Problem Instance (PI) of the scheduling environment.

Data structure of a problem instance

- Let the set $T = \{T_t, \ldots, T_{|T|}\} : \forall\ t \in \{1, \ldots, |T|\}$ denote an arbitrary scheduling period. Given a discrete change in time horizon ΔT_1, we move from T_1 to T_2.
- Let the set $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ denote an n number of jobs, which must be completed during a scheduling period T.
- Let $pr_j \in \mathbb{R}^+$ denote a positive integer number that indicates the priority of job J_j.
- Let the set $f = \{f_g, \ldots, f_{|f|}\} : \forall\ g \in \{1, \ldots, |f|\}$ denote an $|f|$ number of families. Every family f_g comprises a subset of jobs $J_{j,g} \subset J$. Jobs within a family share the same or similar requirements.
- Let the set $D = \{D_{pr}, \ldots, D_{|D|}\} : \forall\ pr \in \{1, \ldots, |D|\}$ denote a $|D|$ number of families. Every family D_{pr} comprises a subset of jobs $J_{j,k} \subset J$. Jobs of a single family share the same priority. In some scheduling environments, we might want to group jobs into two different families according to their priorities and requirements.
- Let the set $O_j = \{O_{o,j}, \ldots, O_{|O|,j}\} : \forall\ o \in \{1, \ldots, |O|\}$ denote an $|O|$ number of operations that compose a job $J_j \in J$. All operations must be completed to process a job fully.
- Let the set $S = \{S_s, \ldots, S_{|S|}\} : \forall\ s \in \{1, \ldots, |S|\}$ denote an $|S|$ number of processing stages. We may omit to define this set if we deal with a single-stage environment.
- Let the set $M = \{M_{i,s}, \ldots, M_{m,s}\} : \forall\ i \in \{1, \ldots, m\}$ denote an m number of machines that are available at every stage $S_s \in S$.
- Let $p_{s,i,j} \in \mathbb{R}^+$ denote the required processing time of a job $J_j \in J$ to be completed by a machine $M_i \in M$ on the processing stage $S_s \in S$. We may omit the s subscript and refer to the processing time of a job by $p_{i,j}$ if we are dealing with a single-stage problem. It is also usual to omit the i subscript of the processing time (p_j) if all machines are identical, resulting in a Pm scheduling environment.
- Let $MS \in \mathbb{R}^+$ denote the overall number of major setup times of all machines M during a scheduling period T. We increase this number subject to the family-dependent setup constraint discussed in Subsection 2.1.2.

- Let $ms_i \in \mathbb{R}^+$ denote the required time to setup a machine M_i when switching between two families.

Supported constraints

MESEAS method supports modeling multiple constraints based on the integrated data model and a considered scheduling environment. One could also extend the available constraints by properly extending the data structure behind the data model. The following points briefly overview the included constraints:

- Release date $\boldsymbol{\beta = r_j}$
- Family dependent setup times $\boldsymbol{\beta = fmls,\ f_{g,h}}$.
- Machine breakdowns $\boldsymbol{\beta = brkdwn}$.
- Machine eligibility restrictions $\boldsymbol{\beta = M_j}$.
- Machine capacity constraint $\boldsymbol{\beta = M^C}$.

Supported objective measures

We will briefly discuss the most important objective values based on our analysis of objective measures for addressing scheduling concerns in Subsection 2.1.3.

- The makespan $\boldsymbol{\gamma = C_{max}}$ and its possible variations.
- The mean flow time $\boldsymbol{\gamma = \overline{F}}$ and its possible variations.
- The number of major setup times $\boldsymbol{\gamma = MS}$.
- The total energy consumption $\boldsymbol{\gamma = E}$.
- The total tardiness $\boldsymbol{\gamma = T}$ and its possible variations.
- The total number of penalties $\boldsymbol{\gamma = U}$.

The previously mentioned objective measures can be investigated either independently or combined based on a considered scheduling environment. Furthermore, we may also investigate any variation of these main objective measures. For instance, we may investigate the minimization of the maximum flow time $\gamma = F_{max}$ based on the average flow time $\gamma = \overline{F}$. After formulating an SP, we seek to formalize the solution of a problem instance PI, which we will discuss in the coming subsection.

Scheduling solution

The scheduling data model expresses an SP that must be solved for some scheduling period $T = \{T_t, \ldots, T_{|T|}\} : \forall\ t \in \{1, \ldots, |T|\}$. Let the set of all feasible solutions for the scheduling problem during a scheduling period T be expressed by the set $\mathbb{X}$.

The scheduling data model can be instantiated to find a solution schedule $X \in \mathbb{X}$, where jobs in $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ are processed using the available machines $M = \{M_{i,s}, \ldots, M_{m,s}\} : \forall\ i \in \{1, \ldots, m\}$ in the corresponding processing stages $S = \{S_s, \ldots, S_{|S|}\} : \forall\ s \in \{1, \ldots, |S|\}$. The solution represents a schedule $X \in \mathbb{X}$ that comprises a set of matrixes relative to the number of stages such that $X = \{X_s, \ldots, X_{|S|}\} \forall\ s \in \{1, \ldots, |S|\}$. Each matrix $X_s \in X$, of the size $(n \times m)$, represents the mapping of n jobs to m machines on the stage S_s and the sequence in which they are processed.

Equation 3.1 exemplifies the shape of the allocation and sequencing solution for the first processing stage $s = 1$. The matrix columns represent the available m machines in the first processing stage and the allocation map of n jobs to these machines. The order of records in the matrix represents the sequence in which n jobs are processed. The sequencing part of the problem may not be relevant if the problem is subject to machine capacity constraints and/or does not involve a sequencing part. Such scheduling problems are very common in cloud environments where a set of virtual machines must be scheduled on a set of available machines with certain capacities.

$$X_s = \begin{bmatrix} x_{111} & x_{121} & \cdots & x_{1m1} \\ x_{112} & x_{122} & \cdots & x_{1m2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{11n} & x_{12n} & \cdots & x_{1mn} \end{bmatrix}, \quad s = 1, 2, \ldots, |S| \tag{3.1}$$

The quality of the pursued solution for the scheduling problem using the presented methodology is measured based on the selected objective measures and considering the operational constraints. Let $\Gamma = \{\gamma_i, \ldots, \gamma_{|\Gamma|}\} : \forall\ i \in \{1, \ldots, |\Gamma|\}$ denote the set of objective measures that the solution $X \in \mathbb{X}$ must minimize. The objective function can be formulated as depicted in Equation 3.2. Let $\omega = \{\omega_i, \ldots, \omega_{|\Gamma|}\} : \forall\ i \in \{1, \ldots, |\Gamma|\}$ denotes a set of weights and bounded by the number of selected objective values in Γ. The objective function can be formulated as a mono-objective function using weights as presented in Equation 3.3.

$$\arg\min_{X \in \mathbb{X}} \Gamma(X) = \arg\min_{X \in \mathbb{X}} [\gamma_i(X), \ldots, \gamma_{|\Gamma|}(X)] : \forall\ i \in \{1, \ldots, |\Gamma|\} \tag{3.2}$$

$$\arg\min_{X \in \mathbb{X}} \Gamma(X) = \arg\min_{X \in \mathbb{X}} \left(\sum_{i=1}^{|\Gamma|} \omega_i \cdot \gamma_i(X) \right) : \sum_{i=1}^{|\Gamma|} \omega_i = 1 \tag{3.3}$$

3.4.3 Definition of MESEAS method to solve a scheduling problem

Based on the presented data structure for scheduling, presented constraints, discussed objective measures of scheduling problem, and the shape of the perused solution, MESEAS methodology can be defined for solving a scheduling problem as presented in Equation 3.4.

$$MESEAS_{(X,\ T)} = \langle SP \mid PI \mid \Gamma \mid Sim \mid HL \mid Opt \vee ML \rangle \tag{3.4}$$

The six-tuple presentation formalizes the definition and later the instantiation of the method. $MESEAS_{(X,\ T)}$ is defined to find a solution schedule $X \in \mathbb{X}$ that minimizes a formulated objective function Γ during a scheduling period T (cf. Equation 3.2 and Equation 3.3). The instantiation of the method depends on the functional requirements that must be satisfied by the functionality layers of the methodology. It means that some elements of the tuple may not be needed. For instance, a simple scheduling solution with a single objective measure can be obtained using the simulation and heuristic library components. The following bullet points interpret the structure of Equation 3.4:

- $SP = \langle \alpha \mid \beta \mid \boldsymbol{\gamma} \rangle$ formalizes the scheduling envrionment that must be considered.
- PI: A problem instance that complies with the definition of the SP and relies on the presented data structure in Section 3.4.2.
- Γ: A set of considered objective values, based on which we seek to find a solution $X \in \mathbb{X}$ that minimizes these values. The considered SP, given the objective measures, may be solved either subject to multi-objective optimization or weighted-sum mono-objective optimization (cf. Section 3.4.2, Equation 3.2, and Equation 3.3).
- Sim: A simulation model that is built based on the structure of the considered scheduling environment SP. It is used as a function to evaluate and construct the final scheduling solution X during a scheduling period T. The simulation model/s are instantiated based on the simulation component of MESEAS methodology, which will be discussed in Subsection 3.4.4.
- HL: An instance of the Heuristic Libaray components (HL), which provides the simulation Sim access to a set of selected allocation and sequencing constructive heuristics during the scheduling period T . The design of HL components, along with some constructive allocation and sequencing heuristics, will be discussed in Section 3.5.
- Opt: An Optimization model (Opt), which utilizes the simulation Sim and the HL components of MESEAS to solve a problem instance PI. The considered PI is solved given a scheduling period T with the objective to minimize Γ . The optimization component of the presented methodology will be thoroughly discussed in Subsection 3.6.2.
- ML: A Machine Learning model (ML), which also relies on the simulation Sim and the HL components of MESEAS to solve a problem instance PI. The ML component of the presented methodology will be elaborated in Subsection 3.6.4.

3.4.4 Simulation component

In the previous section, we discussed the first three elements, SP, PI, and Γ, that are required to instantiate the presented methodology $MESEAS_{(X,\ T)} = \langle \boldsymbol{SP} \mid \boldsymbol{PI} \mid \boldsymbol{\Gamma} \mid \boldsymbol{Sim} \mid HL \mid Opt \ \vee \ ML \rangle$. This section discusses the fourth component in the tuple, namely the

simulation component (Sim). We rely on UML components and sequence diagrams to elaborate on the design and execution of the component, respectively. Figure 3.7 depicts the design of the simulation component in MESEAS methodology using a UML component diagram. The final design of the component allows the modeling of various single-stage and multi-stage scheduling problems in cloud and manufacturing environments.

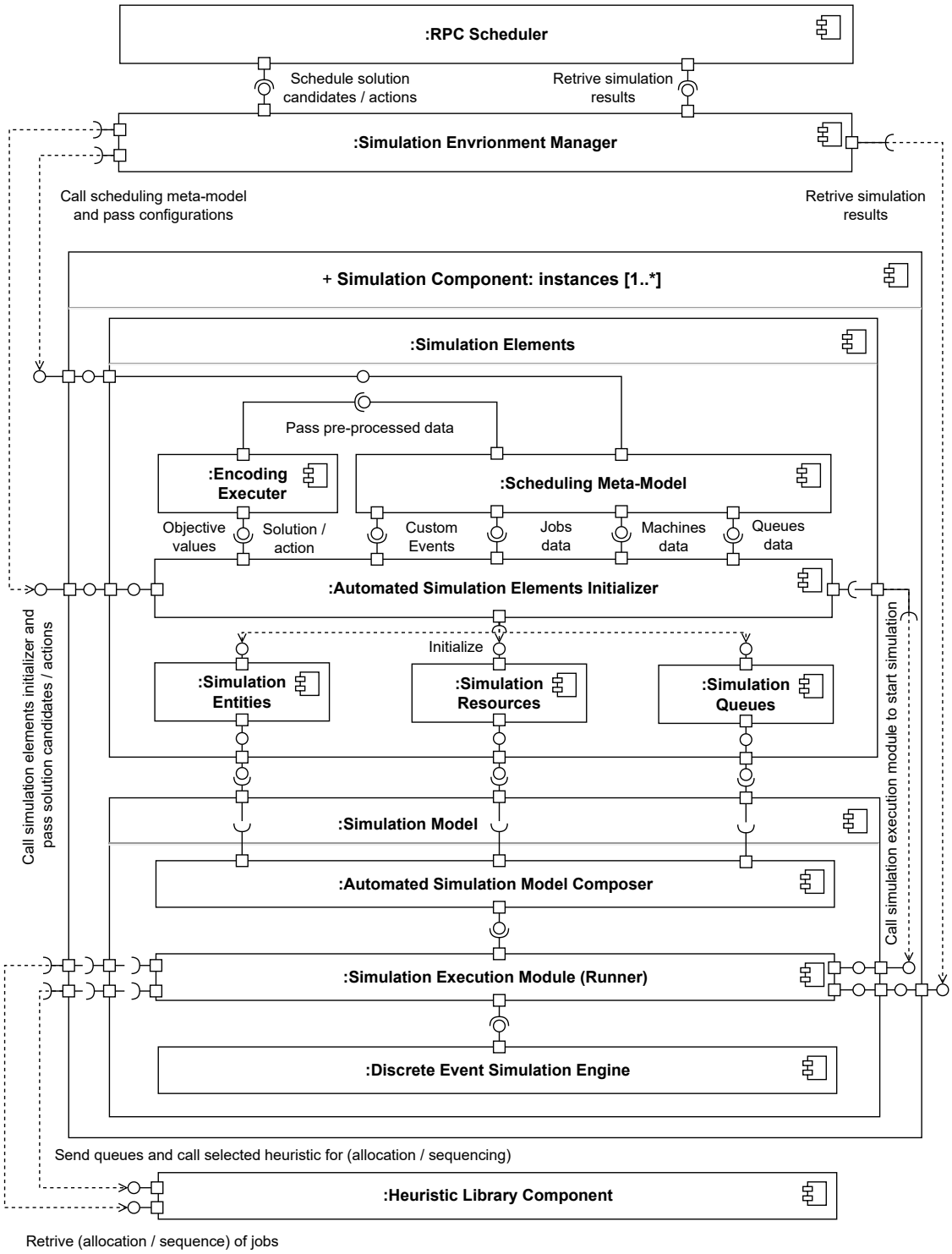


Figure 3.7: UML component diagram of the simulation component.

With automation in mind, we developed the simulation component with automated procedures for simulation building and execution that rely on a previously discussed meta-model for scheduling problems. The simulation component consists of the simulation elements and simulation model sub-components as illustrated in Figure 3.7. The simulation elements component encompasses the main logical modules required to create and initialize a simulation model automatically. The simulation environment manager processes requests and passes the required data to the simulation elements sub-component of the simulation component. Based on the received encoding model and scheduling data model of a considered system, the required number of simulation instances is initialized. The data model for scheduling problems and possibly encoding models includes queues data, machines data, jobs-related data, and custom events in addition to the shape of solution and objective values. These data represent the structural data of a considered scheduling environment that is used by the simulation elements initializer to initialize simulation entities, resources, and queues. After initialization, the simulation elements initializer passes the behavioral data (problem instance) to the simulation execution module of the simulation model component. The runner relies on the automated simulation model composer to connect to the awaiting simulation entities, resources, and queues. The simulation model utilizes a discrete event simulation engine. The execution module starts running a simulation to investigate the possible outcomes of the consumed behavioral data of the considered scheduling environment.

Based on the type of the request, the first two functionality layers of the proposed methodology allow a user to utilize an allocation and/or sequencing constructive heuristic. The execution module of the simulation model sub-component handles the communication between the simulation model component and the heuristic library component. As for the upper functionality layers of the proposed methodology, solution individuals of the optimization component or actions of the machine learning component are passed to the execution module, which communicates with the heuristic library component during a scheduling period. Section 3.5 discusses in length the design of the heuristic library component and how it interacts with the simulation component.

We relied on discrete event simulation cores of the Salabim and CloudSim Plus simulation packages to finalize the development of the simulation component of the presented methodology. The employed simulation packages will be discussed in the implementation overview (cf. Chapter 4). The selection of simulation packages is based on systematic investigation. We collaborated with colleagues to analyze the utilization of open-source simulation software as an alternative for commercial simulation packages in (Lang et al., 2021a). We faced several challenges using propriety simulation packages, such as a lack of parallelization and difficulty in integrating machine learning libraries. We will highlight some issues in the next section (cf. Subsection 3.4.5). The simulation component offers access to the mentioned simulation packages and is designed to integrate further open-source simulation dependencies flexibly.

To fully harness the potential for scalability in modern hardware, we relied on parallelization technologies to process requests for simulation or distribute optimization and

machine learning requests. Both optimization and machine learning components utilize simulation techniques to evaluate the quality of solutions, which can be processed in parallel using available physical resources. Unfortunately, many propriety simulation packages either do not support parallelization or lack the flexibility to be integrated with open-source machine learning techniques. The early prototype of the proposed methodology relied on such a propriety simulation package. To achieve parallelization and scalability objectives, we replicated the entire logical source code of the research artifact and extended it using open-source technologies. The decision to migrate the artifact is based on a systematic analysis.

Figure 3.8 presents a UML sequence diagram that explains the roles, communications, and data exchange between the previously discussed components while executing a simulation run. Based on the request, the simulation environment manager retrieves a scheduling model and the required data to pass them to instantiate a simulation model. The simulation component acknowledges receiving and initializes the simulation model elements sub-component. After the initialization of required simulation elements, a simulation model is composed through the simulation model sub-component. To maintain consistency, in this sequence diagram, it is assumed that the simulation model is always used in conjunction with the heuristic library, which is initialized by the simulation model sub-component (cf. Figure 3.8). After initializing the heuristic library component, an allocation map is requested to map the jobs to available machines in the first processing stage. Finally, simulation execution starts with entering the depicted loop to investigate future scenarios of a considered scheduling environment.

After the allocation map is generated using some allocation algorithm, simulation entities (jobs) are notified with the corresponding allocation to enter the appropriate simulation queues (machine queues or final queues). Every simulation resource (machine) has a set of queued jobs that must be processed considering the objective values. Given the scheduling problem, simulation resources start dispatching simulation entities during the scheduling period using some sequencing algorithm from the heuristic library. After finishing a job $J_j \in J$, a machine $M_i \in M$ retrieves job data in the simulation queue i. After receiving the job data, a sequence is requested before dispatching the next job. The heuristic library component initializes the requested sequencing heuristic and returns a processing sequence to the requesting machine. Based on the sequence, the machine dispatches the first job in the sequence and starts processing it.

After finishing processing the job, the job enters the next queuing stage if multiple processing stages are required. Otherwise, it leaves the system. The recorded events of the simulation are thereby used to update the intermediate schedule, including the starting time and finishing time of the job J_j being completed by machine M_i. The simulation is advanced after all simulation queues are updated and simulation resources are notified of changes. Upon leaving a machine, the machine M_i is idle and available again to process another job. The next job is dispatched and processed similarly. This process is repeated during the simulation until all jobs on all processing stages are completed. Once the simulation is finished, the results are passed back to the simulation environment manager.

The obtained simulation result of the simulation is either pushed back to the user or to the corresponding optimization or machine learning components (cf. Section 3.6). In the next section, the design and development of the heuristic library component will be discussed. The section presents some exemplarily developed allocation and sequencing heuristics, which can be used in conjunction with the simulation component to investigate solutions beyond investigating what-if scenarios.

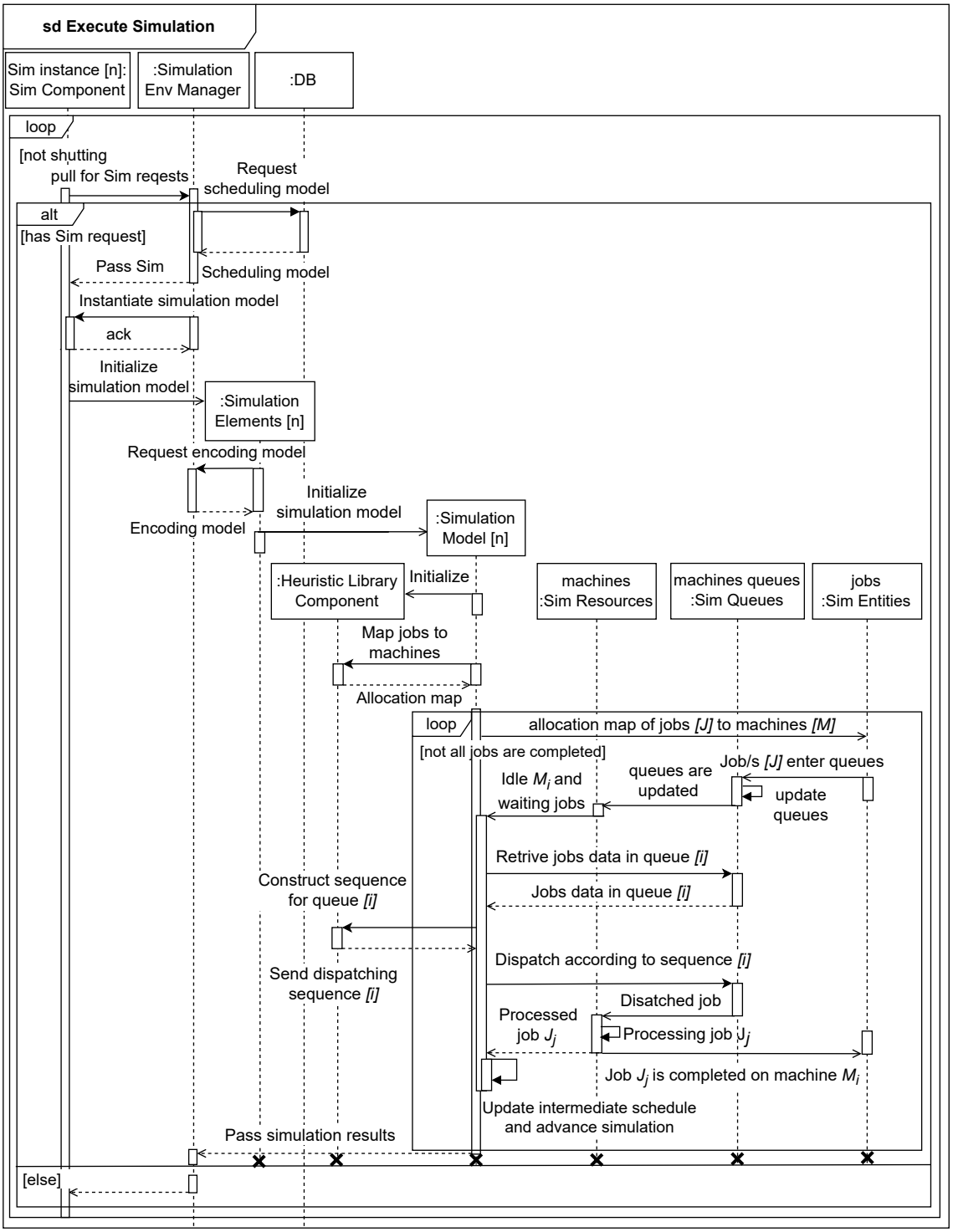


Figure 3.8: UML sequence diagram of the simulation component.

3.4.5 Design evaluation of modeling and simulation components

In the design stage and early prototyping, we conducted multiple investigations on single-stage and multiple-stage scheduling problems in cloud and manufacturing environments. The objectives of the preliminary studies were to validate the designed components, ensure the integrity and flexibility of the data model, and adopt simulation techniques to investigate solutions for scheduling concerns.

In Nahhas et al. (2017a), discrete event simulation methods are investigated to develop an evaluation component for metaheuristics improvement methods to address multi-stage scheduling problems in manufacturing. We relied on the presented mathematical and data models for scheduling to solve a two-stage scheduling problem $\langle HFS_2\ (P_4,\ P_5) \mid f_{g,h} \mid C_{max}, \sum T_j \rangle$. The evaluation is conducted on four real scheduling problem instances. The computational results show that the GA outperforms Simulated Anealing (SA) and Tabu Search (SA) in solving the problems by minimizing the objective values. The research showed that the simulation component is a powerful method for evaluating what-if scenarios for scheduling problems. These findings stayed consistent with a further study, in which we investigated further constraints such as machine breakdowns $\langle HFS_2\ (P_4,\ P_5) \mid f_{g,h}, brkdwn \mid C_{max}, \sum T_j \rangle$ (Aurich et al., 2017). The investigation also shows that adopting population-based improvement methods such as GA techniques guarantees an edge in performance compared to other metaheuristics.

Similarly, we investigated the use of a simulation component and the discussed data model for scheduling to address scheduling concerns in a cloud environment. In Nahhas et al. (2018a), we developed a simulation model to act as an evaluation component for what-if scenarios. Then, it was integrated with constructive heuristic methods and later one with improvement methods (Nahhas et al., 2019a) to study load management of a real cloud environment in which 290 SAP application instances are hosted. The considered scheduling problem was subject to the minimization of energy consumption, taking into account performance constraints $\langle P_m \mid D_{pr}, M^C \mid E, U \rangle$. The simulation experiments showed that applied heuristic methods for scheduling virtual machines can achieve up to 70 % energy-saving during a scheduling period of one week. We relied on workload patterns that were obtained based on interviews with the partner.

However, the computational efforts of both prototypical implementations to achieve a solution for scheduling problems could be improved. In both studies, we utilized propriety discrete event simulation, which at the time did not support parallelization. That, obviously, significantly impacts the required computational effort. In addition, automation of system modeling with a different number of processing stages and machines was very challenging and always required additional manual effort to maintain the data model. Therefore, as discussed in the design of the simulation component, we briefly discussed the migration to an open-source discrete event simulation engine in the final design of the simulation component (cf. Subsection 3.4.4). In addition to the adoption of discrete event simulation methods to analyze the data model, we collected initial insights into the design of the heuristic library components.

3.5 Heuristics library components

In the previous section, we discussed the first four elements, SP, PI, Γ, and Sim that are required to instantiate the presented methodology $MESEAS_{(X,\ T)} = \langle \boldsymbol{SP} \mid \boldsymbol{PI} \mid \boldsymbol{\Gamma} \mid \boldsymbol{Sim} \mid \boldsymbol{HL} \mid Opt \vee ML \rangle$. Defining the first four elements in the tuple provides access to the functionalities of the bottom layers of the presented methodology. It facilitates investigating what-if scenarios in some considered scheduling environments. This section explains the design of the fifth element in the discussed tuple presentation of the methodology, which is the Heuristic Library components (HL). To sustain clarity in explaining the design of the HL component, we utilize the pseudocode convention to discuss the rationale of some of the developed construction allocation and sequencing heuristics. We also utilize UML sequence diagrams to elaborate on the overall design and execution of the HL component.

We developed and adopted various constructive heuristics and PDRs while designing the MESEAS heuristics library components, which fulfill the discussed requirements specification of the second functionality layer. Section 2.1 thoroughly discussed scheduling problems. To reduce the complexity of a scheduling problem, it is a common practice to deal with its underlying allocation part independently from its sequencing part of the problem. Therefore, the heuristic layer of the presented methodology provides access to developed allocation and sequencing heuristic algorithms. For consistency, some of the developed allocation and sequencing algorithms will be discussed using pseudocode in the coming subsections. The adopted constructive heuristics from related works will be omitted and referenced in the corresponding papers.

3.5.1 Allocation constructive heuristics

Family-Increasing Workload-Increasing (FI-WI): Following a similar convention, we will assume some parallel machine scheduling problem before describing an algorithm. Based on the Shortest Processing Time (SPT) PDR and the First Fit Increasing (FFI), we derived an allocation algorithm that combines the logic of both algorithms for family group scheduling. Algorithm 7 presents the logical design of the allocation algorithm for solving parallel machines scheduling problems Qm. The Family-Increasing Workload-Increasing ($FI-WI$) constructs an allocation schedule following a combination of the FFI and SPT algorithms. Let $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ be a set of jobs, which must be scheduled on the set of machines $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$. The workload of the machines is denoted by a set $l = \{l_i, \ldots, l_m\} : \forall\ i \in \{1, \ldots, m\}$ such that every machine M_i is associated with its current workload l_i. The considered problem is subject to the family operation constraint (cf. Section 2.1.2). It implies that the set $f = \{f_g, \ldots, f_{|f|}\} : \forall\ g \in \{1, \ldots, |f|\}$ denote an $|f|$ number of families. Jobs are grouped into these families. Every family f_g is associated with the sum of the processing time of all jobs p, resulting in f_g^p. The $FI-WI$ heuristic starts with computing the overall processing time of families f_g^p (cf. Algorithm 7, lines 4-9).

Based on the overall computed processing time, families are sorted in ascending order (cf. Algorithm 7, line 10). Finally, the algorithm always starts with the shortest family

in terms of processing time and schedules its jobs on the least loaded machine (cf. Algorithm 7, lines 11-17). After scheduling all jobs within a family, the scheduled family is removed, the workload of machines is updated, and the machines are sorted again in terms of workload in ascending order (cf. Algorithm 7, lines 18-20). This process is repeated until all jobs are scheduled on the machine and an allocation solution is returned (cf. Algorithm 7, lines 11-22). We developed this algorithm to favor system efficiency objective values such as the minimization of the C_{max} (Baker and Trietsch, 2009, p. 234), $\overline{F}$, or total number of major setup times $MS \in \mathbb{R}^+$.

In the description of all coming algorithms, we omit details of some functions to sustain clarity. For instance, sorting functions such as (sortAscendingByAccumulatedProcessingTime) or (SortAscendingByWorkload) are invoked without discussing their logic. Similarly, we do not detail simple functions such as (getFirstFamily), (getFirstMachine), (constructAllocationMap), (removeScheduledFamily), or (updateWorkload) and only invoke them in the pseudocode of Algorithm 7.

Algorithm 7 Family-Increasing Workload-Increasing ($FI-WI$) - Allocation

```
    /* We sort ascending in terms of accumulated processing time of jobs in families f^p.
    Then we sort ascending in terms of machines workload l_i. */
 1: procedure FI-WI (f, J, M)
 2:     Z_allocation ← ∅
 3:     M_sorted ← sortAscendingByWorkload (M, index = l_i)
 4:     for each f_g ∈ f do
 5:         f_g^p ← NULL                                          ▷ Initialization
 6:         for each J_j ∈ f_g do
 7:             f_g^p ← f_g^p + p_j          ▷ Compute the overall processing time in families
 8:         end foreach
 9:     end foreach
10:     f_sorted ← sortAscendingByAccumulatedProcessingTime (f, index = f^p)
11:     while f_sorted ≠ ∅ do
12:         f_g ← getFirstFamily (f_sorted)
13:         M_i ← getFirstMachine (M_sorted)
14:         for each J_j ∈ f_g do
15:             scheduleJobs (J_j, M_i)
16:             Z_allocation ← constructAllocationMap (J_j, M_i)
17:         end foreach
18:         f_sorted ← removeScheduledFamily (f_sorted, f_g)
19:         M ← updateWorkload (M, M_i, f_g^p)
20:         M_sorted ← SortAscendingByWorkload (M, index = l_i)
21:     end while
22:     return Z_allocation
23: end procedure
```

Family-Decreasing Workload-Increasing (FD-WI): Similarly, the Longest Processing Time (LPT) PDR and the First Fit Increasing (FFI) are combined to derive the logic of the $FD-WI$ algorithm. It can also be utilized for group scheduling, considering family constraints. Algorithm 8 depicts the logical structure of the allocation algorithm for solving parallel machines scheduling problems Qm. The Family-Decreasing Workload-Increasing ($FD-WI$) builds an allocation schedule following a combination of the LPT and FFI algorithms. Given parallel machines scheduling problem where we must schedule a set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ on the set of machines $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$. The workload of the machines is denoted by a set $l = \{l_i, \ldots, l_m\} : \forall\ i \in \{1, \ldots, m\}$ such that every machine M_i is associated with its current workload l_i. This formulation also considers family constraint (cf. Section 2.1.2). Jobs are grouped into families resulting in a set $f = \{f_g, \ldots, f_{|f|}\} : \forall\ g \in \{1, \ldots, |f|\}$. Every family f_g is associated with the sum of the processing time of all jobs p and denoted by f_g^p.

Algorithm 8 Family-Decreasing Workload-Increasing ($FD-WI$) - Allocation

```
/* We sort descending in terms of accumulated processing time of jobs in families f^p.
   Then we sort ascending in terms of machines workload l_i. */
1:  procedure FD-WI (f, J, M)
2:      Z_allocation ← ∅
3:      M_sorted ← sortAscendingByWorkload (M, index = l_i)
4:      for each f_g ∈ f do
5:          f_g^p ← NULL                                   ▷ Initialization
6:          for each J_j ∈ f_g do
7:              f_g^p ← f_g^p + p_j        ▷ Compute the overall processing time in families
8:          end foreach
9:      end foreach
10:     f_sorted ← sortDescendingByAccumulatedProcessingTime (f, index = f^p)
11:     while f_sorted ≠ ∅ do
12:         f_g ← getFirstFamily (f_sorted)
13:         M_i ← getFirstMachine (M_sorted)
14:         for each J_j ∈ f_g do
15:             scheduleJobs (J_j, M_i)
16:             Z_allocation ← constructAllocationMap (J_j, M_i)
17:         end foreach
18:         f_sorted ← removeScheduledFamily (f_sorted, f_g)
19:         M ← updateWorkload (M, M_i, f_g^p)
20:         M_sorted ← SortAscendingByWorkload (M, index = l_i)
21:     end while
22:     return Z_allocation
23: end procedure
```

The $FD-WI$ heuristic starts with computing the overall processing time of families f_g^p (cf. Algorithm 8, lines 4-9). Afterward, families are sorted in, however, descending order by overall workload (cf. Algorithm 8, line 10). Based on the sorted list, the algorithm starts with the longest family in terms of processing time and schedules its jobs on the least loaded machine (cf. Algorithm 8, lines 11-17). Similarly, after scheduling jobs of a family, the scheduled family is removed, the workloads of machines are updated, and machines are sorted again in terms of workload in ascending order (cf. Algorithm 8, lines 18-20). This process is repeated until all jobs are scheduled on the machine and an allocation map is returned (cf. Algorithm 8, lines 11-22). This algorithm also favors system efficiency objective values such as the minimization of the total number of major setup times $MS \in \mathbb{R}^+$. However, it might lead to an inferior makespan C_{max}.

Deadline-Aware Family Fit Increasing (DA-FFI): Based on the logical design of previous algorithms, many additional algorithms are inferred. For instance, Algorithm 9 presents the rational of the *DA-FFI*, which is largely based on the logical design of the *FI-WI* in Algorithm 7. In this algorithm, we combined the SPT, the FI-WI, and the Earliest-Due-Date (EDD) algorithm. The DA-FFI considers the deadlines of jobs within families before constructing allocation solutions. Given an environment where parallel machines are available to process jobs Qm. It is desired to allocate the set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ to be processed by the set of machines $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$. The workloads of the machines are also computed given some scheduling period and can be expressed by the set $l = \{l_i, \ldots, l_m\} : \forall\ i \in \{1, \ldots, m\}$. Every machine M_i is associated with its current workload l_i during the scheduling period.

Jobs are grouped into families that are denoted by the set $f = \{f_g, \ldots, f_{|f|}\} : \forall\ g \in \{1, \ldots, |f|\}$. Every family f_g is associated with the sum of the processing time of all jobs p, and the earliest due date of a job in it d_j. The overall processing time of a family f_g is expressed by f_g^p, and the earliest due date of a job is expressed by f_g^d. The $DA-FFI$ heuristic starts with computing the overall processing time of every family f_g^p and identifying the earliest due date of a job in every family f_g^d (cf. Algorithm 9, lines 4-13). Based on the overall computed processing time, families are sorted in ascending order (cf. Algorithm 9, line 14) and then sorted again in ascending order by the earliest due date (cf. Algorithm 9, line 15). The algorithm starts with the family that contains the job with the highest priority. In the case of multiple families with the highest priority, it selects the family with the shortest overall processing time. Then, it schedules its jobs on the least loaded machine (cf. Algorithm 9, lines 17-19). After scheduling all jobs within a family, the scheduled family is removed, the workload of machines is updated, and the machines are sorted again in terms of workload in ascending order (cf. Algorithm 7, lines 23-25). This process is repeated until all families with their associated jobs are scheduled on the machine, and an allocation solution is returned (cf. Algorithm 7, lines 16-27). We developed this algorithm to favor more system efficiency objective values such as the minimization of the C_{max} or total number of major setup times $MS \in \mathbb{R}^+$. The algorithm also considers customer satisfaction concerns, such as minimizing the total tardiness T.

Algorithm 9 Deadline-Aware Family-Fit-Increasing ($DA-FFI$) - Allocation

/* We sort ascending in terms of accumulated processing time of jobs in families f^p, followed by the earliest deadline of jobs f^d within families. Finally, we sort ascending in terms of machines workload l_i. */

1: **procedure** DA-FFI (f, J, M)
2: $Z_{allocation} \leftarrow \emptyset$
3: $M_{sorted} \leftarrow$ sortAscendingByWorkload (M, $index = l_i$)
4: **for each** $f_g \in f$ **do**
5: $f_g^p \leftarrow NULL$ ▷ The superscript p denotes the processing time of all jobs in f_g
6: $f_g^d \leftarrow MAX$ ▷ The superscript d denotes the earliest due date in f_g
7: **for each** $J_j \in f_g$ **do**
8: $f_g^p \leftarrow f_g^p + p_j$ ▷ Compute the overall processing time in families
9: **if** $f_g^d > d_j$ **then**
10: $f_g^d \leftarrow d_j$ ▷ Associate the family with the earliest due date of a job in it.
11: **end if**
12: **end foreach**
13: **end foreach**
14: $f_{sorted} \leftarrow$ sortAscendingByAccumulatedProcessingTime (f, $index = f^p$)
15: $f_{sorted} \leftarrow$ sortAscendingByDueDate (f_{sorted}, $index = f^d$)
16: **while** $f_{sorted} \neq \emptyset$ **do**
17: $f_g \leftarrow$ getFirstFamily (f_{sorted})
18: $M_i \leftarrow$ getFirstMachine (M_{sorted})
19: **for each** $J_j \in f_g$ **do**
20: scheduleJobs (J_j, M_i)
21: $Z_{allocation} \leftarrow$ constructAllocationMap (J_j, M_i)
22: **end foreach**
23: $f_{sorted} \leftarrow$ removeScheduledFamily (f_{sorted}, f_g)
24: $M \leftarrow$ updateWorkload (M, M_i, f_g^p)
25: $M_{sorted} \leftarrow$ SortAscendingByWorkload (M, $index = l_i$)
26: **end while**
27: **return** $Z_{allocation}$
28: **end procedure**

Deadline-Workload-Aware Family-Fit-Increasing (DWA-FFI): We developed the *DWA-FFI* algorithm based on the rationale of the *DA-FFI* with a stronger focus on customer satisfaction objective values such as the total number of penalties U and the total tardiness T. Algorithm 10 depicts a pseudocode of the algorithm for solving parallel machine scheduling problems Qm. Shortly, in this environment, the set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ must be processed by the set of machines $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$. The workloads of the machines are denoted by the set $l = \{l_i, \ldots, l_m\} : \forall\ i \in \{1, \ldots, m\}$. Jobs are grouped into families, which are expressed by the set $f = \{f_g, \ldots, f_{|f|}\} : \forall\ g \in$

$\{1, \ldots, |f|\}$. During a scheduling period, we designed the algorithm to distribute jobs with the earliest due dates between the machines as uniformly as possible, given some critical parts of the scheduling period. In essence, the algorithm relies on a Critical Horizon (CH) parameter to achieve this objective. The CH parameter defines the critical part of a scheduling period. Assuming that our scheduling period is a month with jobs that must be completed within this scheduling period. The CH parameter can be set to a week. In essence, all jobs with due dates falling within the first week of the scheduling period are within the Critical Horizon parameter.

Algorithm 10 Deadline-Workload-Aware Family-Fit-Increasing ($DWA-FFI$)

/* We sort ascending in terms of accumulated processing time of prioritized jobs in families f^{ch}, followed by the earliest deadline of jobs within a family f^d. */

1: **procedure** DWA-FFI (f, J, M, CH)
2: $Z_{allocation} \leftarrow \emptyset$
3: $M_{sorted} \leftarrow$ sortAscendingByWorkload (M, $index = l_i$)
4: **for each** $f_g \in f$ **do**
5: $f_g^p \leftarrow NULL$
6: $f_g^d \leftarrow MAX$
7: **for each** $J_j \in f_g$ **do**
8: **if** $d_j \leq CH$ **then** ▷ Workload within the critical horizon.
9: $f_g^{ch} \leftarrow f_g^{ch} + p_j$
10: **end if**
11: **if** $f_g^d > d_j$ **then**
12: $f_g^d \leftarrow d_j$ ▷ Associate the family with the earliest due date of a job in it.
13: **end if**
14: **end foreach**
15: **end foreach**
16: $f_{sorted} \leftarrow$ sortAscendingByAccumulatedProcessingTime (f, $index = f^{ch}$)
17: $f_{sorted} \leftarrow$ sortAscendingByDueDate (f_{sorted}, $index = f^d$)
18: **while** $f_{sorted} \neq \emptyset$ **do**
19: $f_g \leftarrow$ getFirstFamily (f_{sorted})
20: $M_i \leftarrow$ getFirstMachine (M_{sorted})
21: **for each** $J_j \in f_g$ **do**
22: scheduleJobs (J_j, M_i)
23: $Z_{allocation} \leftarrow$ constructAllocationMap (J_j, M_i)
24: **end foreach**
25: $f_{sorted} \leftarrow$ removeScheduledFamily (f_{sorted}, f_g)
26: $M \leftarrow$ updateWorkload (M, M_i, f_g^p)
27: $M_{sorted} \leftarrow$ SortAscendingByWorkload (M, $index = l_i$)
28: **end while**
29: **return** $Z_{allocation}$
30: **end procedure**

The $DWA-FFI$ heuristic starts with computing the processing time of jobs, which must be delivered within the critical horizon for every family f_g^p (cf. Algorithm 10, lines 4-10). Within the same sub-procedure, the algorithm identifies the earliest due date of a job in every family f_g^d (cf. Algorithm 10, lines 11-15). Then, families are sorted in ascending order first by the computed processing time of critical jobs and then the earliest due date (cf. Algorithm 10, lines 16-17).

The algorithm starts with the family that contains the job with the highest priority. In the case of multiple families with the highest priority, it selects the family with the shortest accumulated processing time and schedules its jobs on the least loaded machine (cf. Algorithm 10, lines 21-24). After scheduling, the selected family is removed, the workload of machines is updated, and the machines are sorted again in terms of workload in ascending order (cf. Algorithm 10, lines 25-27). This process is repeated until all families with their associated jobs are scheduled on the machine, and an allocation solution is returned (cf. Algorithm 10, lines 18-29). The CH makes the algorithm aware of the criticality of the workload being distributed between machines. In turn, it strives for an equal distribution of critical jobs in terms of due dates between machines. This rationale contributes to minimizing the total tardiness T and the number of penalties U. It still takes into account system efficiency since jobs are scheduled in groups to avoid excessive reconfigurations of machines to minimize the total number of major setup times $MS \in \mathbb{R}^+$.

Energy-Aware Family-Fit-Decreasing (EA-FFD): The last allocation algorithm is an example of an energy-aware algorithm, which is developed to allocate jobs to machines subject to capacity and priority constraints (cf. Subsection 2.1.2). Given some scheduling environment, a set of jobs (virtual machines, J^R) must be scheduled on a set of machines (computing servers, M^C). The set of jobs is denoted by $J^R = \left\{J_j^R, \ldots, J_n^R\right\} : \forall\, j \in \{1, \ldots, n\}$ and set of available parallel machines is expressed by $M^C = \left\{M_i^C, \ldots, M_m^C\right\} : \forall\, i \in \{1, \ldots, m\}$. An arbitrary job $J_j^R \in J^R$ can be allocated to a machine $M_i^C \in M^C$ only if the machine can satisfy all its resources requirements subject to the constraints discussed in ($\vdash \Upsilon$, Equation 2.1). In this use case, jobs are usually grouped into families in terms of their priorities, which are expressed by the set $D^R = \left\{D_{pr}^R, \ldots, D_{|D|}^R\right\} : \forall\ pr \in \{1, \ldots, |D|\}$. Every family D_{pr}^R comprises a subset of jobs $J_{j,q}^R \subset J^R$. In many scheduling environments in the cloud and manufacturing, it is a common practice to prioritize jobs of certain types of customers to avoid violation of Service Level Agreements (SLAs) or delivery dates. Algorithm 11 depicts a pseudocode of the discussed energy-aware algorithm that takes into account various priority classes while scheduling jobs to machines. The algorithm is developed to minimize the overall energy consumption $\overline{E}$ during a scheduling period T while minimizing the violation of priority classes U.

The rationale of the algorithm is designed to loop over multiple job- and machine-related data with the objective of finding a suitable allocation. The algorithm starts by sorting priority family groups in terms of their pr in ascending order (cf. Algorithm 11, Line 3) before starting the first loop. Then, the family $D_{(j,k)}^R$ with the highest priority is selected for scheduling (i.e., the family with the smallest pr). Jobs within a priority

Algorithm 11 Energy-Aware Family-Fit-Decreasing ($EA-FFD$) - Allocation

```
/* Equation 2.1 - We sort according to the memory requirement in line 7 */
1: procedure EA-FFD (D^R, J^R, M^C, M_idle)
2:     Z_allocation ← ∅
3:     D^R_sorted ← sortAscendingByPriority (D^R_pr, index = pr)
4:     while D^R_sorted ≠ ∅ do
5:         D^R_(j,k) ← getFirstFamily (D^R_sorted)
6:         R ← getJobRequirments (D^R_(j,k))        ▷ Extract the set of job requirements
7:         J^R_sorted (j,k) ← sortDescendingByRequirement (D^R_(j,k), index = R_ram)
8:         while J^R_sorted (j,k) ≠ ∅ do
9:             Ē_j^min ← MAX                ▷ The minimal energy consumption of a job as max
10:            M^C_i,j ← NULL               ▷ The job is not scheduled on any machine yet
11:            satisfied ← False            ▷ Boolean to check requirements are met
12:            J^R_j ← getFirstJob (J^R_sorted (j,k))
13:            R ← getJobRequirments (J^R_j)        ▷ Extract the set of job requirements
14:            for each M^C_i ∈ M^C do
15:                C ← getMachineCapacities (M^C_i)
16:                for each (R_r ∈ R) and (C_c ∈ C) do
17:                    if R_r ≤ C_c then        ▷ Machine capacity satisfies job's requirement?
18:                        satisfied ← True
19:                    else
20:                        satisfied ← False
21:                    end if
22:                end for
23:                if satisfied == Ture then
24:                    Ē_j ← estimatePowerConsumption (J^R_j, M^C_i)
25:                    if Ē_j < Ē_j^min then
26:                        M^C_i,j ← M^C_i          ▷ Pick the machine with lower energy
27:                        Ē_j^min ← Ē_j            ▷ update the lowest energy
28:                    end if
29:                end if
30:            end for
31:            if M^C_i,j ≠ NULL then
32:                Z_allocation ← constructSolution (J^R_j, M^C_i,j)
33:                J^R_sorted (j,k) ← remove (J^R_sorted (j,k), J^R_j)
34:                M^C ← updateWorkload (M^C, M^C_i,j, J^R_j)
35:            else        ▷ Activate sleeping machine to satisfy the requirements of a job
36:                M^C_i,j ← activateIdelMachine (M_idle)
37:                Z_allocation ← constructSolution (J^R_j, M^C_i,j)
38:                J^R_sorted (j,k) ← remove (J^R_sorted (j,k), J^R_j)
39:                M^C ← updateWorkload (M^C, M^C_i,j, J^R_j)
40:            end if
41:        end while
42:        D^R_sorted ← removeFamily (D^R_sorted, J^R_sorted (j,k))    ▷ Remove allocated family
43:    end while
44:    return Z_allocation
45: end procedure
```

family have certain requirements denoted by the superscript R. Following the original First-Fit-Decreasing algorithm, we sort the jobs within a single family in descending order in terms of requirement. In the considered use case, we might tolerate oversubscription in terms of CPU but not in terms of memory. The algorithm sorts jobs according to memory requirements (cf. Algorithm 11, Line 7). After sorting jobs within the selected family, the algorithm enters its first nested loop to schedule every job on available machines (cf. Algorithm 11, Lines 8-41). The algorithm depends on some initialization steps every time a job is selected (cf. Algorithm 11, Lines 9-13).

After selecting a job, the algorithm enters its second nested loop that is designed to schedule jobs to machines with minimal energy consumption (cf. Algorithm 11, Lines 14-30). Here, we loop over every machine and check whether it satisfies the requirements of the selected job (cf. Algorithm 11, Lines 16-22). If the machine satisfies the requirements of a selected job, the algorithm checks the incurred fraction of energy consumption caused by scheduling the job to the machine (cf. Algorithm 11, Line 24). If the current allocation incurs less energy, the allocation is updated, and the new machine is set to be the destination of the job (cf. Algorithm 11, Lines 25-28). After looping over all machines, the algorithms check whether at least one machine is found to allocate the job (cf. Algorithm 11, Lines 31-40). If a suitable machine is found, the algorithm schedules the job, removes it from the current family, and updates the workload of the machines (cf. Algorithm 11, Lines 31-34). Otherwise, the algorithm activates an idle machine, allocates the job to this machine, updates the workload, and submits the activated machine to the set of available machines (cf. Algorithm 11, Lines 35-40). Here, for the sack of readability, another loop over the set of idle machines to find the machine, which incurs a minimal fraction of energy consumption to process the job, is committed. After finding a proper allocation of the selected job, the algorithm starts over with the next job (cf. Algorithm 11, Line 8) until all jobs are already scheduled. After scheduling all jobs within the selected family, the next family is selected back in the upper loop (cf. Algorithm 11, Line 4).

3.5.2 Sequencing constructive heuristics

Depending on the scheduling environment, finding an efficient solution may imply dealing with only allocation, which is a very common case in managing workloads in cloud environments. However, it may involve dealing with the allocation followed by the sequencing part of the problem, which is often encountered in manufacturing. Therefore, the heuristic constructive heuristic layer of the developed methodology also contains a set of developed and adopted sequencing algorithms.

EDD-Family Shortest Processing Time (EDD-FSPT): The $EDD-FSPT$ algorithm combines the use of the EDD and SPT priority dispatching rules. It is also developed to function for group sequencing, considering family setup time constraints (cf. Section 2.1.2). The algorithm is developed with a stronger emphasis on system efficiency to minimize the total number of major setup times $MS \in \mathbb{R}^+$ while attempting somewhat to re-

duce total tardiness T and penalties U. Given a scheduling problem in which jobs are already allocated to machines. For every machine $M_i \in M$, a subset of jobs $\hat{J} \subset J$ must be completed in some sequence to optimize relevant objective values. The set of machines is denoted by $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$, and the set of jobs by $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$.

Algorithm 12 EDD-Family Shortest Processing Time ($EDD - FSPT$) - Sequencing

/* We sort ascending in terms of accumulated processing time of jobs in families f^p, followed by the earliest deadline of jobs d_j within families. Then, we construct a processing sequence.*/

procedure EDD-FSPT ($\hat{f}$, $\hat{J}$, M_i)
  $Z_{sequence} \leftarrow \emptyset$
  $f_{(EDD-SPT)} \leftarrow \emptyset$ ▷ The shortest family with the earliest due date of a job in it.
  for each $f_g \in \hat{f}$ **do** ▷ $\hat{f}$ is a set of the current families allocated to the machine M_i
    $f_g^p \leftarrow NULL$ ▷ Initialization.
    $f_g^d \leftarrow MAX$ ▷ Initialization.
    for each $J_j \in f_g$ **do**
      $f_g^p \leftarrow f_g^p + p_j$ ▷ Compute the overall processing time in families.
      if $f_g^d > d_j$ **then**
        $f_g^d \leftarrow d_j$ ▷ Associate the family with the earliest due date of a job in it.
      end if
    end foreach
  end foreach
  $f_{sorted} \leftarrow$ sortAscendingByAccumulatedProcessingTime ($\hat{f}$, $index = f^p$)
  $f_{sorted} \leftarrow$ sortAscendingByDueDate (f_{sorted}, $index = f^d$)
  while $f_{sorted} \neq \emptyset$ **do**
    $f_g \leftarrow$ getFirstFamily (f_{sorted})
    $f_g \leftarrow$ sortAscendingByDueDate (f_g, $index = d_j$)
    if $f_{(EDD-SPT)} == \emptyset$ **then**
      $f_{(EDD-SPT)} \leftarrow f_g$
    end if
    for each $J_j \in f_g$ **do**
      scheduleJobs (J_j, M_i)
      $Z_{sequence} \leftarrow$ constructSequenceMap (J_j, M_i)
    end foreach
    $f_{sorted} \leftarrow$ removeScheduledFamily (f_{sorted}, f_g)
  end while
  return $Z_{sequence}$, $f_{(EDD-SPT)}$
  We will need the $f_{(EDD-SPT)}$ in another algorithm.
end procedure

Jobs may belong to different families in terms of their priority or their operational nature. It implies that on every machine M_i, a subset of families $\hat{f} \subset f$ with their associated jobs are queued. Based on preliminaries, the set of families is expressed by $f = \{f_g, \ldots, f_{|f|}\} : \forall\ g \in \{1, \ldots, |f|\}$. Splitting the allocation part of the scheduling problem from its sequencing part results in m number of single machine sequencing problems that must be solved. For every machine M_i, the algorithm starts with accumulating the overall processing time of every family $f_g \in \hat{f}$ (cf. Algorithm 12, lines 4-13). Inside the same loop, the algorithm associates every family with the earliest due date overall jobs (cf. Algorithm 12, lines 9-11).

Then, the algorithm sorts prepared families ascending by the accumulated processing time f^p followed ascending by the earliest due date f^d (cf. Algorithm 12, lines 14-15). Finally, we start sequencing jobs by selecting the first family and then sorting its jobs by the due date (cf. Algorithm 12, lines 17-18). This combination results by dispatching the smallest family with the EDD job among all families and then sorting it by due dates of jobs $f_{(EDD-SPT)}$. This selection is returned in line 20 since it is needed in another algorithm (cf. Algorithm 12, line 20). After sorting the family, the algorithm starts constructing the processing sequence for every job in the family (cf. Algorithm 12, lines 22-25). Finally, the sequenced family is removed before starting again with the next family back in line 16. This process is repeated until all families and their jobs are sequenced, and a sequence map is returned (cf. Algorithm 12, lines 16-28).

EDD-Family Longest-Processing-Time (EDD-FLPT): The EDD-FLPT design combines the utilization of the EDD and LPT to construct a scheduling sequence of jobs on a machine M_i. Algorithm 13 depicts the pseudocode of this heuristic. The algorithm is largely similar to Algorithm 12 in terms of initialization, preparation, and sequencing. The soul and yet significant difference between these heuristics lies in the sorting mechanism.

The EDD-FLPT sorts families in descending order by the accumulated processing time of jobs in them before selecting the next family (Algorithm 12, line 14). This, in turn, results in completely different solutions. The algorithm yields a sequence where $MS = |f|$. The rationale of both algorithms starts by selecting a family and then keeps sequencing jobs within the same family till all jobs are completed. As a result, machines are reconfigured $|f|$ times over the entire scheduling period, favoring significantly the minimization of the total number of major setup times $MS = |f|$. The MS here would be the minimum possible value. The selection of the shortest family among all families contributes significantly to minimizing the makespan C_{max}, especially in multi-stage environments.

Algorithm 13 EDD-Family Longest-Processing-Time ($EDD - FLPT$) - Sequencing

/* We sort descending in terms of accumulated processing time of jobs in families f^p, followed by the earliest due date of jobs d_j within families. Then, we construct a processing sequence.*/

procedure EDD-FLPT ($\hat{f}$, $\hat{J}$, M_i)
  $Z_{sequence} \leftarrow \emptyset$
  $f_{(EDD-LPT)} \leftarrow \emptyset$ ▷ The longest family with the earliest due date of a job in it.
  for each $f_g \in \hat{f}$ **do** ▷ $\hat{f}$ is a set of the current families allocated to the machine M_i
    $f_g^p \leftarrow NULL$ ▷ Initialization.
    $f_g^d \leftarrow MAX$ ▷ Initialization.
    for each $J_j \in f_g$ **do**
      $f_g^p \leftarrow f_g^p + p_j$ ▷ Compute the overall processing time in families.
      if $f_g^d > d_j$ **then**
        $f_g^d \leftarrow d_j$ ▷ Associate the family with the earliest due date of a job in it.
      end if
    end foreach
  end foreach
  $f_{sorted} \leftarrow$ sortDescendingByAccumulatedProcessingTime ($\hat{f}$, $index = f^p$)
  $f_{sorted} \leftarrow$ sortAscendingByDueDate (f_{sorted}, $index = f^d$) ▷ Consider deadline
  while $f_{sorted} \neq \emptyset$ **do**
    $f_g \leftarrow$ getFirstFamily (f_{sorted})
    $f_g \leftarrow$ sortAscendingByDueDate (f_g, $index = d_j$)
    if $f_{(EDD-LPT)} == \emptyset$ **then**
      $f_{(EDD-LPT)} \leftarrow f_g$
    end if
    for each $J_j \in f_g$ **do**
      scheduleJobs (J_j, M_i)
      $Z_{sequence} \leftarrow$ constructSequenceMap (J_j, M_i)
    end foreach
    $f_{sorted} \leftarrow$ removeScheduledFamily (f_{sorted}, f_g)
  end while
  return $Z_{sequence}$, $f_{(EDD-LPT)}$
  We will need the $f_{(EDD-LPT)}$ in another algorithm.
end procedure

Deadline-Aware Sequencing-Increasing (DA-SI): The logical design of the *DA-SI* consists of two layers: family sequencing and job sequencing. The family sequencing part is straightforward and utilizes the EDD-FSPT presented in Algorithm 12. The second layer is rather more complex and involves up to three nested loops designed to avoid violating job due dates. Algorithm 14 depicts the overall rationale of the developed algorithm using pseudocode convention. The rationale of the algorithm relies on grouping jobs into families subject to two dimensions in terms of their part type and priority, yielding two family sets $\hat{f}$ and $\hat{D}$, respectively.

The algorithm is developed to underline customer satisfaction objective measures to minimize the total tardiness T and penalties U while attempting to reduce the total

number of major setup times MS and the makespan C_{max}. The algorithm produces a sequence for processing jobs that are allocated to a machine. After allocation, each machine $M_i \in M$ is utilized to process a subset of jobs $\hat{J} \subset J$ in some order considering the minimization of objective values. The set of machines is denoted by $M = \{M_i, \ldots, M_m\} : \forall\ i \in \{1, \ldots, m\}$, and the set of jobs by $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$. Jobs are grouped into families in terms of priority and part types, yielding two family types. For every machine M_i, a subset of part-types families $\hat{f} \subset f$ with their associated jobs are queued. The set of families is denoted by $f = \{f_g, \ldots, f_{|f|}\} : \forall\ g \in \{1, \ldots, |f|\}$. Queued jobs in each machine also belong to another family type $\hat{D} \subset D$ in terms of their priorities pr. The priority families are expressed by the set $D = \{D_{pr}, \ldots, D_{|D|}\} : \forall\ pr \in \{1, \ldots, |D|\}$ denote a $|D|$ number of families.

The logical design of the algorithm can be divided into three logical blocks. The first block starts in line 4 by selecting the smallest family that contains a job with the highest priority using the EDD-FSPT (cf. Algorithm 14, line 4). The selected family is sorted ascending by the due date of jobs d_j. Then, the first job is scheduled in the sequence, a machine setup time is added to the overall sequence, and the scheduled job is removed (cf. Algorithm 14, lines 5-8). The second block starts in line 9 and goes until line 20. Before scheduling a job $J_j \in J$, the algorithm is designed to iterate over all priority families $\hat{D}$ on a machine M_i (cf. Algorithm 14, lines 12-20). If the priority pr_j of a job J_j is lower than a family priority D_{pr}, the algorithm iterates overall jobs in this family to compute the overall processing time of higher priority jobs, resulting in an inner loop (cf. Algorithm 14, lines 16-18).

The third logical block is designed to decide to either schedule the selected job or switch to another family to avoid due date violations and associated tardiness (cf. Algorithm 14, lines 21-43). First, the algorithm sorts families with higher priorities in an ascending order by priority. After initialization, it checks for every family whether scheduling the job might cause a potential violation or not (cf. Algorithm 14, lines 27-38). Based on the computed estimation, if the job does not cause potential tardiness, it is scheduled, added to the sequence map, and removed from the associated sets (cf. Algorithm 14, lines 39-42). Otherwise, scheduling fails, and the algorithm starts over in line 4 to check the next job in the family.

In essence, the algorithm is designed to keep scheduling jobs in the same family while attempting to avoid violating the due dates of other jobs with higher priorities in other families (cf. Algorithm 14, lines 9-43). After scheduling all possible jobs in the selected family, the algorithm starts again in line 3 to select the next priority family. This process is repeated until all jobs in $\hat{J} \subset J$ and their associated setup times are scheduled. Finally, the procedure ends by returning a sequence of jobs $Z_{sequence}$ to be executed for their completion on the machine M_i (Algorithm 14, line 45). The presented rationale in Algorithm 14 is simplified for single-stage scheduling problems to maintain consistency. Multi-stage sequencing is relatively more complicated since the processing times of jobs in further stages must be considered. Usually, heuristic approximation is applied to estimate the waiting times of jobs and possibly their processing time in further processing stages.

Algorithm 14 Deadline-Aware Sequencing-Increasing ($DA-SI$) - Sequencing

```
/* This algorithm rely on various sorting of families D̂ and f̂ for sequencing jobs. */
procedure DA-SI (D̂, f̂, Ĵ, M_i, ms_i)
    Z_sequence ← ∅
    while Ĵ ≠ ∅ do
        f_(EDD−SPT) ← EDD-FSPT(f̂, Ĵ, M_i)        ▷ Here we want to get f_(EDD−SPT)
        f_sorted ← sortAscendingByDueDate (f_(EDD−SPT), index = d_j)
        Z_sequence ← constructSequenceMap (J_j, M_i) ▷ Start with the first family.
        Z_sequence ← addSetupToSequenceMap (ms_i, M_i)
        D̂, f̂, Ĵ ← removeScheduledJob (D̂, f̂, Ĵ, J_j)
        for each J_j ∈ f_sorted do
            D_higher ← ∅
            violateDueDateHigherPriorityJobs ← True
            for each D_pr ∈ D̂ do        ▷ D_pr denotes a family of jobs sharing priority.
                if pr_j > D_pr then        ▷ D_pr Smaller priority implies higher priority.
                    D^p_pr ← 0
                    D_higher ← addHigherPriorityFamily(D_pr)
                    for each J_k ∈ D_pr do
                        D^p_pr ← D^p_pr + p_k ▷ Compute the processing time of every family.
                    end foreach
                end if
            end foreach
            if D_higher == ∅ then
                violateDueDateHigherPriorityJobs ← False
            else
                D_higher ← sortAscendingByPriority (D_higher, index = D_pr)
                numSetup = 0
                overallProcessingTime = 0
                for each D^p_pr ∈ D_higher do
                    overallProcessing = overallProcessing + D^p_pr
                    numSetup = numSetup + 1
                    processingAndSetup = overallProcessing + (numSetup ∗ ms_i)
                    C_j = C(Z_sequence) + p_j
                    if (C_j + processingAndSetup) < D^d_pr then
                        violateDueDateHigherPriorityJobs ← False
                    else
                        violateDueDateHigherPriorityJobs ← True
                    end if
                end for
            end if
            if violateDueDateHigherPriorityJobs == False then
                Z_sequence ← constructSequenceMap (J_j, M_i)
                D̂, f̂, Ĵ ← removeScheduledJob (D̂, f̂, Ĵ, J_j)
            end if
        end for
    end while
    return Z_sequence
end procedure
```

Simulation methods can be applied to utilize the previously discussed algorithms. Combining simulation techniques with constructive allocation and sequencing heuristics facilitates investigating simple, instant solutions for a given scheduling problem. Figure 3.9 depicts a UML sequence diagram that describes the main interactions and lifeline of the major components in the simulation and heuristic library layers of MESEAS methodology.

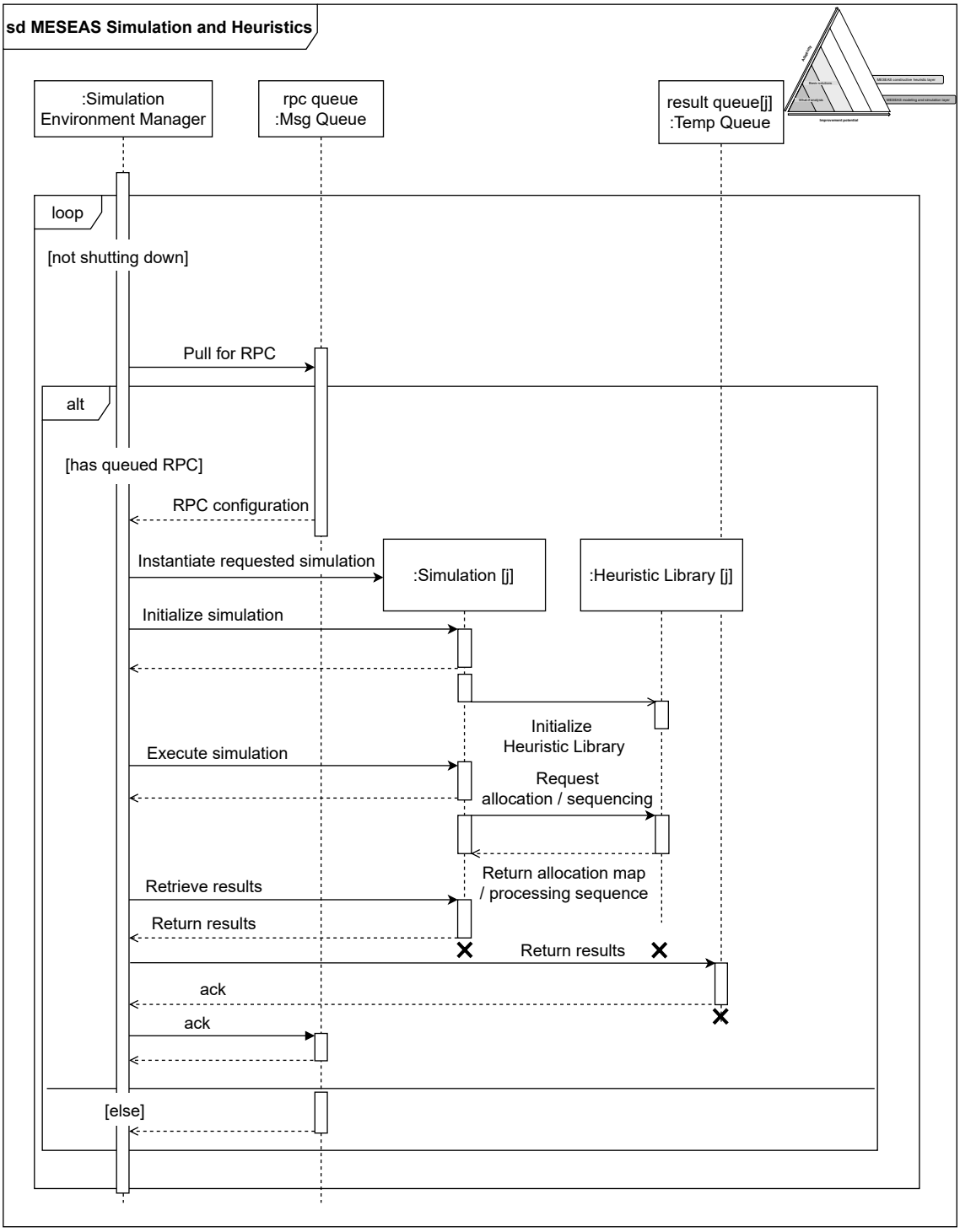


Figure 3.9: UML sequence diagram of the simulation and heuristic layers.

In conclusion, the first two functionality layers of MESEAS methodology are grouped into a single subsystem ≪*MESEAS Simulation and Heuristic Subsystem*≫ as discussed in Figure 3.4. In Figure 3.5, we discussed, using a sequence diagram, how the overall structure of the proposed method functions using a UML sequence diagram. Figure 3.9 UML sequence diagram details how requests to the components of the first two layers of the proposed method are processed. Incoming requests are first processed by the *Simulation Environment Manager*. It is responsible for instantiating and initializing simulation and heuristic library components for a given problem, as depicted in the figure. After initialization, a simulated model is created to investigate what-if scenarios.

If heuristic solutions are also present in the request, corresponding configurations are passed to the simulation component. The simulation component executes the simulation model, which interacts with the heuristic library component. Based on the selected allocation and sequencing heuristic, the simulation model requests a sequence and/or an allocation map for a given set of jobs that must be scheduled. The appropriate algorithm processes the request and returns the results to the simulation component. After simulating a scheduling period, the results of the simulation are pushed back to the user or other components through the environment manager. Here, it is important to emphasize that the optimization and the machine learning components also send requests to the simulation layer and rely heavily on the first two functionality layers of the proposed methodology. As can be noticed in the diagram, *Simulation Environment Manager* constantly awaits requests from the remote call procedure, which we discussed in the MESEAS component diagram and MESEAS sequence diagram.

3.5.3 Design evaluation of heuristic library components

Similarly to the modeling and simulation components, the design of the heuristic library components is evaluated during the early prototyping and implementation stage. The algorithms discussed in previous sections are exemplary allocation and sequencing algorithms, which are presented for parallel machines scheduling problems to maintain consistency. The components of the heuristic library are evaluated in our intermediate results that were published during the research project of this thesis.

In Nahhas et al. (2018a), we investigated a single-stage parallel machine scheduling problem with capacity constraint $\langle P_m \mid D_{pr}, M^C \mid E, U \rangle$. The objective was to minimize the overall energy consumption while considering performance concerns. In that research, we investigated, for instance, the rationale of Family-Decreasing Workload-Increasing (cf. Algorithm 8) and the Family-Decreasing Workload-Increasing (cf. Algorithm 7) algorithms. In Nahhas et al. (2019a), we further evaluated the design of the Energy-Aware Family-Fit-Decreasing (cf. Algorithm 11) and studied the combination of constructive heuristic methods and improvement methods for addressing scheduling problems in cloud environments. In both studies, we relied on simulation techniques for evaluation and investigation of what-if scenarios using heuristics.

In Nahhas et al. (2017a), we evaluated various variations of the discussed sequencing algorithms such as the EDD-Family Shortest Processing Time (cf. Algorithm 12), the EDD-Family Longest Processing Time (cf. Algorithm 13), and other PDRs. The considered multi-stage scheduling problem was solved to minimize the makespan and the total tardiness $\langle HFS_4\ (P_4, P_5) \mid f_{g,h} \mid C_{max}, \sum T_j \rangle$. We further investigated various variations of the allocation and sequencing algorithms, for instance, the Deadline-Aware Sequencing-Increasing (cf. Algorithm 14) in Nahhas et al. (2018b). The investigated problem included two additional processing stages of the same considered manufacturing environment in the field of printed circuit board production $\langle HFS_4\ (P_4, P_5, 1, 1) \mid f_{g,h}, brkdwn \mid C_{max}, \sum T_j \rangle$. In this paper, we started the initial investigation of combining heuristic and improvement methods in conjunction with simulation techniques for addressing scheduling problems subject to machine breakdowns. The findings in this research and a follow-up research on scheduling in cloud environments Nahhas et al. (2019b) highlighted the need for more computationally efficient techniques to address scheduling concerns.

The results of these investigations, combined with analysis of related works, suggested that combining constructive heuristics and improvement can deliver high-quality solutions for scheduling concerns with acceptable computational effort. The hybrid combination can mitigate, to some extent, the problem of computational efficiency. The remaining open question was how to design the optimization component to seamlessly integrate it with various allocation and sequencing heuristics for addressing scheduling problems. This question was only partially answered due to the significant role of the simulation component, which was developed using a propriety simulation package. As a result, integrating and testing different improvement methods was challenging. In addition, we could not achieve the required level of automation, which resulted in high manual effort to maintain the integrated methods. Nonetheless, the initial observations and partial evaluation were fundamental for finalizing the design of the optimization component, which we will discuss in the coming subsection.

3.6 Optimization and machine learning components

Usually, if the desired solutions for scheduling problems are subject to multiple objective values, constructive heuristics may not be sufficient. In addition, adaptivity requires automation of solution generation to adhere to changes in the considered environment and facilitate rapid decision-making processes. Therefore, the upper functionality layers of the proposed method facilitate obtaining improved solutions for scheduling concerns in the considered environment using the optimization component. Adaptive solutions can be achieved after sufficient training of Deep Reinforcement Learning (DRL) technique using the machine learning component. The requests of the upper functionality layer of the developed methodology are handled by the *≪MESEAS Optimization and Machine Learning Subsystem≫*. This subsystem encompasses two core functional components: Optimization and Machine Learning components. In Figure 3.5, we discussed how the overall architecture of the MESEAS methodology functions and highlighted the layer managers,

which forward requests to the optimization and machine learning subsystem of the presented methodology. Figure 3.10 depicts a UML sequence diagram that describes the interactions between various components in this subsystem and how incoming requests are processed in general before discussing in greater detail the main components.

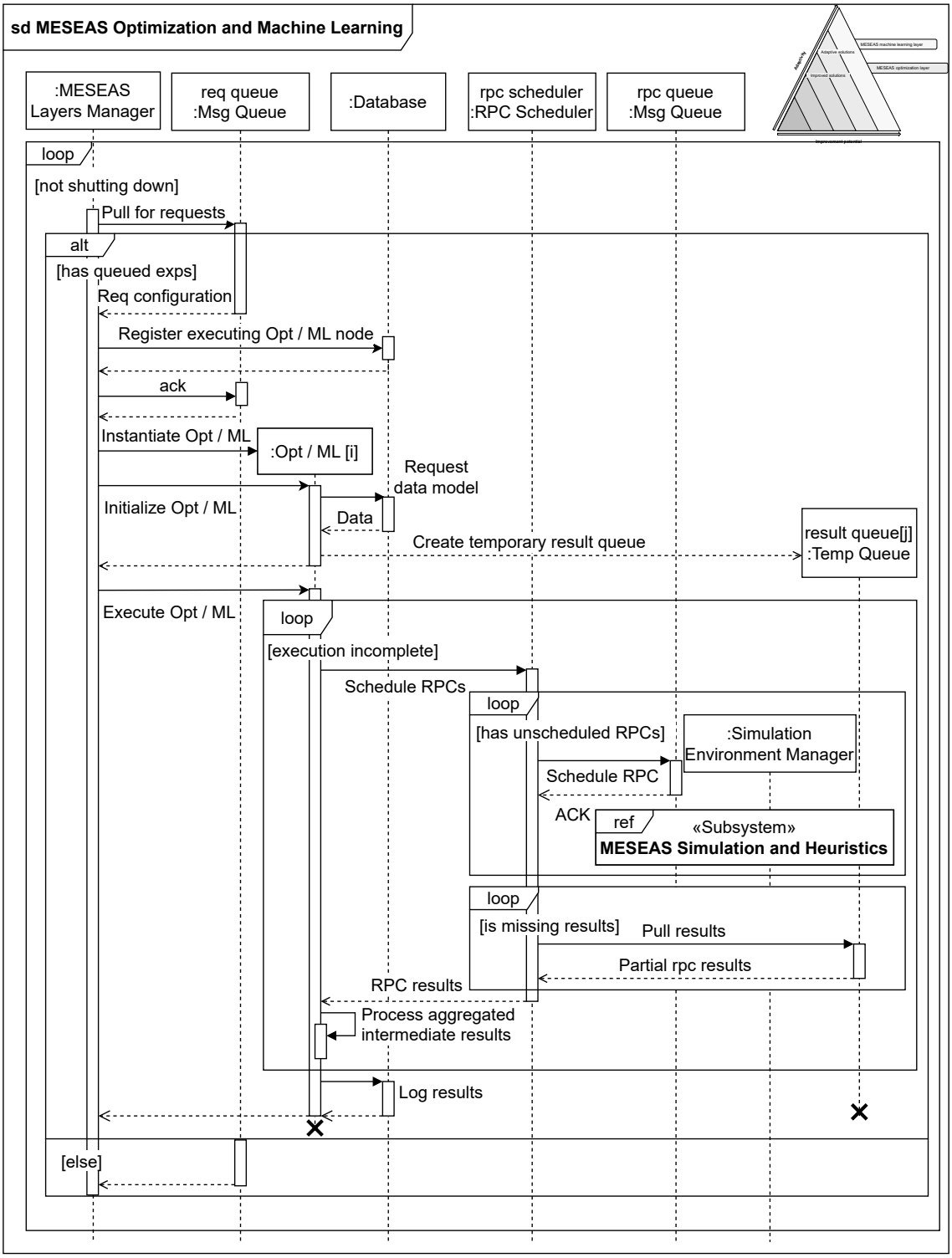


Figure 3.10: UML sequence diagram of the optimization and machine learning layers.

Starting with the MESEAS layers manager, which keeps processing incoming requests in its online state. Upon receiving corresponding requests, it instantiates and initializes desired instances of optimization or a machine-learning component. Based on the received request, the initialized component retrieves the corresponding scheduling data model from the database. The data model includes structural and behavioral data, which are required to initiate and initialize the optimization, machine learning, and simulation components. Then, it starts execution modules to trigger optimization or machine learning runs. We discussed earlier that the optimization and machine learning component depends on the simulation and heuristic subsystem for solutions evaluation and/or generation. Depending on the type of request, solutions, individuals, or actions are scheduled by the Remote Procedure Call (RPC). It handles the communication and the synchronization of retrieved results from the simulation and heuristic subsystem using the simulation environment manager, which we discussed in the previous section. As long as the optimization is not finished or the training is not finalized, simulation requests are queued and processed by the simulation subsystem. After completion, the intermediate and aggregated results are processed and stored in the database to be exposed to the user. If some intermediate results are missing, requests are pushed again to the simulation and heuristic subsystem to ensure the integrity of the optimization or machine learning training processes. For instance, the fitness values of some individuals of the current generation are missing.

To meet the requirement specifications of the upper functionality layers of MESEAS, we developed evolutionary optimization and DRL encoding models to adopt these methods for solving scheduling problems. After defining the first five elements of $MESEAS_{(X,\ T)} = \langle SP \mid PI \mid \Gamma \mid Sim \mid HL \mid Opt \ \vee \ ML \rangle$, we introduce the adoption of evolutionary optimization methods Opt in Subsection 3.6.1 and the adoption of DRL methods ML in Subsection 3.6.3. The developed and adopted models are then integrated into the overall architecture of MESEAS and detailed using UML component diagrams in Subsection 3.6.2 and Subsection 3.6.4. Finally, the functionality of every component is elaborated using UML sequence diagrams in the mentioned subsections.

3.6.1 Optimization encoding models for scheduling problems

In genetics, a *genotype* is an abstract representation of a set of genes that are imprinted in an individual of some population (Stansfield, 1991, pp. 24-25). The entire set of genes represents a chromosome (Stansfield, 1991, pp. 1). The expression and translation of these genes in a certain environment produce a set of distinctive and measurable characteristics called in genetic *phenotype* (Stansfield, 1991, p. 24). In evolutionary optimization, the translation of the genotype (problem encoding) to produce the characteristics (phenotype) of a solution for an optimization problem is obviously simplified and abstracted from natural process (Ronald, 1997, p. 44). An optimization problem is usually represented using a direct or indirect encoding (Talbi, 2009, pp. 41-42). In direct encoding, the genotype is translated directly to the phenotype without a decoder. In contrast, indirect encoding would require a decoder to translate the genotype and express the phenotype (Talbi, 2009,

p. 41). In scheduling, it is more common to rely on an indirect encoding of the problem due to complexity and applied constraints (Talbi, 2009, p. 41). For example, sequencing and ordering problems can be encoded using integer values (Talbi, 2009, p. 35), resulting in a *"permutation encoding"* (Ronald, 1997, p. 44). To address allocation problems, a linear representation of discrete values may be adapted to map jobs to machines (Talbi, 2009, p. 36).

Indirect discrete encoding based on attributes markers: In the proposed method, this encoding model is adopted and used to deal with the allocation part of scheduling problems. The genotypes are represented using discrete integer values. Some attributes of jobs can be used as markers to develop the encoding. For instance, we rely on the family-type (e.g., f_g, D_{pr}) features of jobs to encode the problem (cf. Subsection 3.4.1). The genotypes are grouped into a vector of variable size to form the chromosome. The chromosome is expressed by the vector Φ^{attr} as presented in Equation 3.5. The size of the vector corresponds to the size of the used marker $|f|$ in a considered problem instance. Each genotype Φ_φ represents the allocation of a family f_g that comprises a set of jobs $J_{j,g} \subset J$ to a machine $M_i \in M$ on the corresponding processing stage.

$$\Phi^{attr} = \Big[\Phi_\varphi, \quad \cdots, \quad \Phi_{|f|}\Big] : \quad (\varphi = 1, 2, \ldots, |f|) \wedge (\Phi_\varphi = 1, 2, \ldots, m) \tag{3.5}$$

Indirect discrete encoding based on allocation heuristic markers: A scheduling problem is solved subject to a scheduling period $T = \{T_t, \ldots, T_{|T|}\} : \forall\ t \in \{1, \ldots, |T|\}$. Based on the design of the proposed methodology, we rely on the discrete event simulation method to apply the scheduling data model and simulate the scheduling period. This encoding is developed by dividing the scheduling period into decision points and applying heuristic markers. In every decision point, the utilized constructive allocation heuristic is changed, and a new allocation map is generated. This encoding model integrates the optimization, the constructive heuristic, and the simulation functionality layers of the proposed methodology (cf. Subsection 3.1.3 and Figure 3.1).

Let $A = \{A_a, \ldots, A_{|A|}\} : \forall\ a \in \{1, \ldots, |A|\}$ denote a set of a number of constructive allocation heuristics available in the heuristic library components of MESEAS. Every element A_a denotes some allocation heuristics in the heuristic library component (cf. Section 3.5). The genotypes are represented using discrete integer values and rely on the allocation heuristics as markers for the encoding. The chromosome shape is represented by a vector that combines the genotypes. Equation 3.6 presents the shape of a solution individual Φ^A using this encoding model. The size of the vector depends on the number of decision points during a scheduling period $T = \{T_t, \ldots, T_{|T|}\} : \forall\ t \in \{1, \ldots, |T|\}$. Each genotype Φ_φ indexes an allocation constructive algorithm A_a, which can be used at the decision point T_φ.

$$\Phi^A = \Big[\Phi_\varphi, \quad \cdots, \quad \Phi_{|T|}\Big] : \quad (\varphi = 1, 2, \ldots, |T|) \wedge (\Phi_\varphi = 1, 2, \ldots, |A|) \tag{3.6}$$

Multi-population encoding based on allocation and sequencing heuristic markers: Following the same principles, this encoding model extends the previous one and depends on the concept of decision points over a scheduling period $T = \{T_t, \ldots, T_{|T|}\} : \forall\ t \in \{1, \ldots, |T|\}$. At each decision point, we can utilize different constructive allocation and sequencing heuristics. This encoding further incorporates the sequencing heuristic component of MESEAS. It exposes the available sequencing heuristics to be utilized by the simulation component to generate improved solutions for scheduling problems (cf. Subsection 3.1.3 and Figure 3.1).

Let $B = \{B_b, \ldots, B_{|B|}\} : \forall\ b \in \{1, \ldots, |B|\}$ denotes a set of b number of constructive sequencing heuristics available in the heuristic library components of the presented methodology. Each B_b represents a sequencing heuristic, which is integrated into the heuristic library component (cf. Section 3.5). An indirect discrete encoding is used to represent the genotypes that are grouped to shape the chromosome of a sequencing solution individual. Equation 3.7 presents the shape of a sequencing solution individual Φ^B using this encoding model. The size of the vector depends on the number of decision points during a scheduling period $T = \{T_t, \ldots, T_{|T|}\} : \forall\ t \in \{1, \ldots, |T|\}$. Each genotype Φ_φ encodes a discrete integer value marking an available sequencing constructive algorithm, which can be used at the decision point T_t.

$$\Phi^B = \Big[\Phi_\varphi, \quad \cdots, \quad \Phi_{|T|}\Big] : \quad (\varphi = 1, 2, \ldots, |T|) \wedge (\Phi_\varphi = 1, 2, \ldots, |B|) \tag{3.7}$$

The overall encoding model combines the allocation Φ^A and sequencing Φ^B representation. It relies on two interdependent populations to construct the final solution of a scheduling problem. The evolutionary operators, including selection, crossover, and mutation, are applied to two independent populations of solution individuals. This encoding results in a cooperative coevolutionary implementation of multiple populations. To evaluate the fitness of an allocation solution individual, the sequencing solution individual must be evaluated at the same time. The decoding and evaluation of both solution individuals yield the construction of the overall solution of a scheduling problem. The multi-population encoding model can be expressed by the set $\Phi = \{\Phi^A,\ \Phi^B\} :$ *subject to Equation* 3.8.

$$\begin{aligned} \Phi^A &= \Big[\Phi_\varphi, \quad \cdots, \quad \Phi_{|T|}\Big] : \quad (\varphi = 1, 2, \ldots, |T|) \wedge (\Phi_\varphi = 1, 2, \ldots, |A|) \wedge \Phi^A \in \Phi \\ \Phi^B &= \Big[\Phi_\varphi, \quad \cdots, \quad \Phi_{|T|}\Big] : \quad (\varphi = 1, 2, \ldots, |T|) \wedge (\Phi_\varphi = 1, 2, \ldots, |B|) \wedge \Phi^B \in \Phi \end{aligned} \tag{3.8}$$

3.6.2 Design of the optimization component

The discussed encoding optimization models are integrated into the overall architecture of MESEAS following modular design. Figure 3.11 presents the structure of the optimization component using a UML component diagram. The optimization component of MESEAS consists of the optimization model and the evaluation model subcomponents. It orchestrates multiple optimization instances based on the received configuration from MESEAS layers manager. The optimization model subcomponent comprises several logical modules,

such as the scheduling model, the optimization model initializer, the algorithm initializer, and the execution module. The scheduling model is designed to pre-process relevant data from the received scheduling data model, which is used to configure the optimization algorithm appropriately. Based on the received scheduling problem, encoding model, and pursued objective values, the optimization model initializer instantiates evolutionary algorithm operators. Meanwhile, it also triggers the evaluation model subcomponent to initialize and execute the evaluation model component. The evaluation model acts as a logical interface between the optimization model and the simulation subsystem.

We developed the main components of MESEAS methodology following modular design principles to achieve flexibility. For instance, we may utilize various simulation engines, which rely on Java or Python dependencies for different scheduling problems. As depicted in Figure 3.11, the optimization provides different evolutionary algorithms such as classical GA originally discussed by Holland (1975) and Non-dominating Sorting Genetic Algorithm three (NSGA III) originally presented by Deb and Jain (2014), and a multi-population variation of the NSGA III. We previously discussed that it is common to solve the allocation and the sequencing part of a scheduling problem independently (Nahhas et al., 2022a). For instance, we adopted the NSGA III to deal with the allocation and sequencing subproblems using two independent populations.

The modular design facilitates flexible integration and customization of additional evolutionary algorithms by implementing reference algorithms and developing the desired encoding model. The optimization model is developed to be utilized by users with different knowledge levels in optimization. Experts can, for instance, pass configuration to customize the selection operator, cross-over function, and mutation function of the selected evolutionary algorithm. If basic configurations are passed, the optimization model relies on our default operators. After initializing the optimization model, an instance of the selected algorithm is initialized with either weighted-sum or muti-objective modes. The algorithm also defaults to multi-objective if the user does not supplement weights to the selected objective values. After initializing the optimization model and desired optimization algorithm, the optimization component relies on the execution module to start executing optimization as presented in Figure 3.11.

The optimization layer of MESEAS methodology utilizes parallelization techniques for multi-optimization experiments since evolutionary algorithms are eventually stochastic optimization techniques. The adopted client-server architecture of the optimization model and the evaluation model ensures the distribution of evaluating candidate solutions between available physical resources, which are running simulation instances. This architecture is fundamental to achieving scalability of physical resources and possibly relying on cloud resources, if necessary, for critical scheduling concerns.

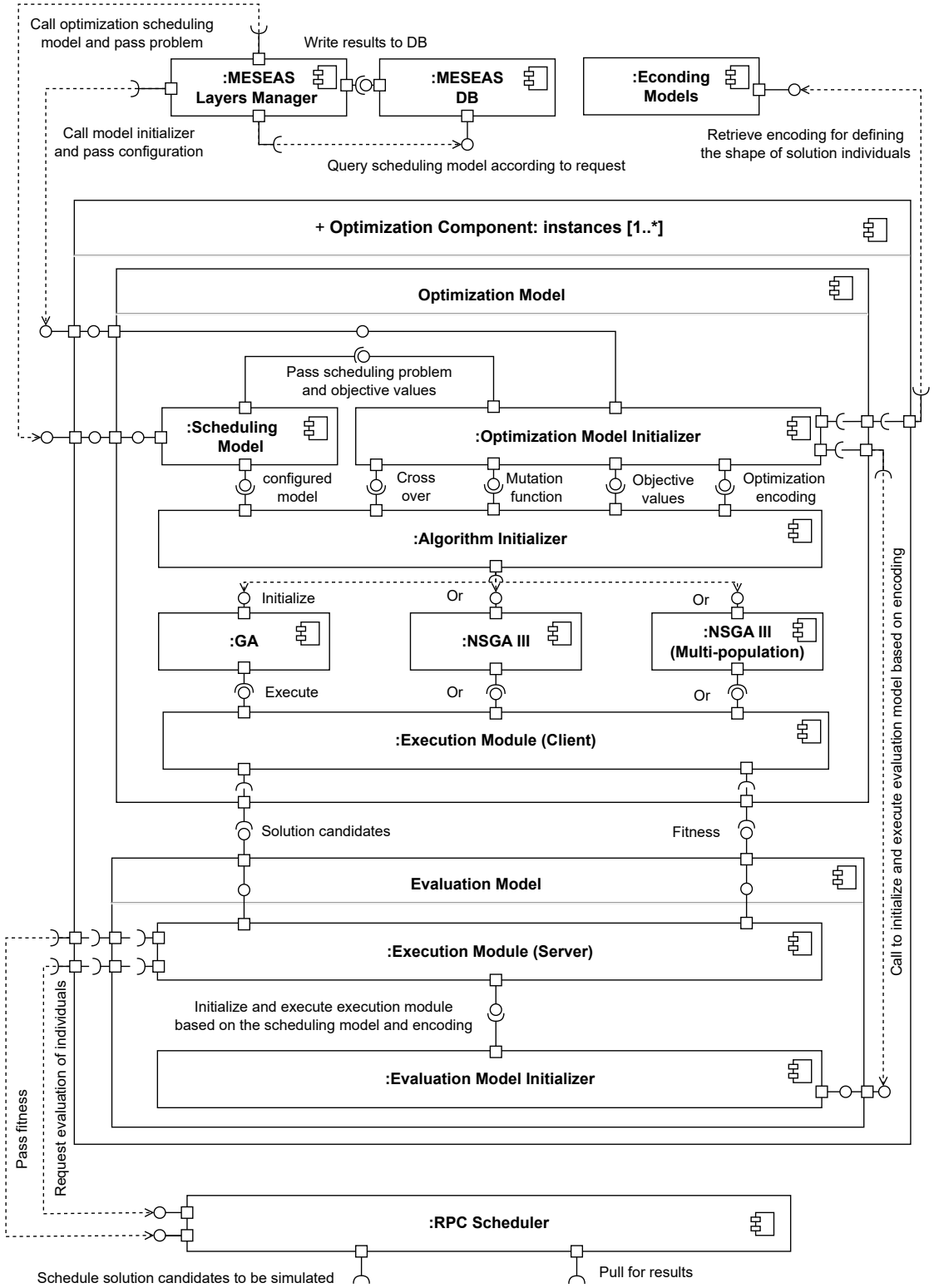


Figure 3.11: UML component diagram of the optimization component.

Figure 3.12 illustrates the rationale of the optimization component during execution using a UML sequence diagram. The presented figure shows the lifeline, exchanged data, and the overall logic of the most significant modules of the optimization component. As long as the optimization instances are online, they await a request from the MESEAS layers manager to start new optimization.

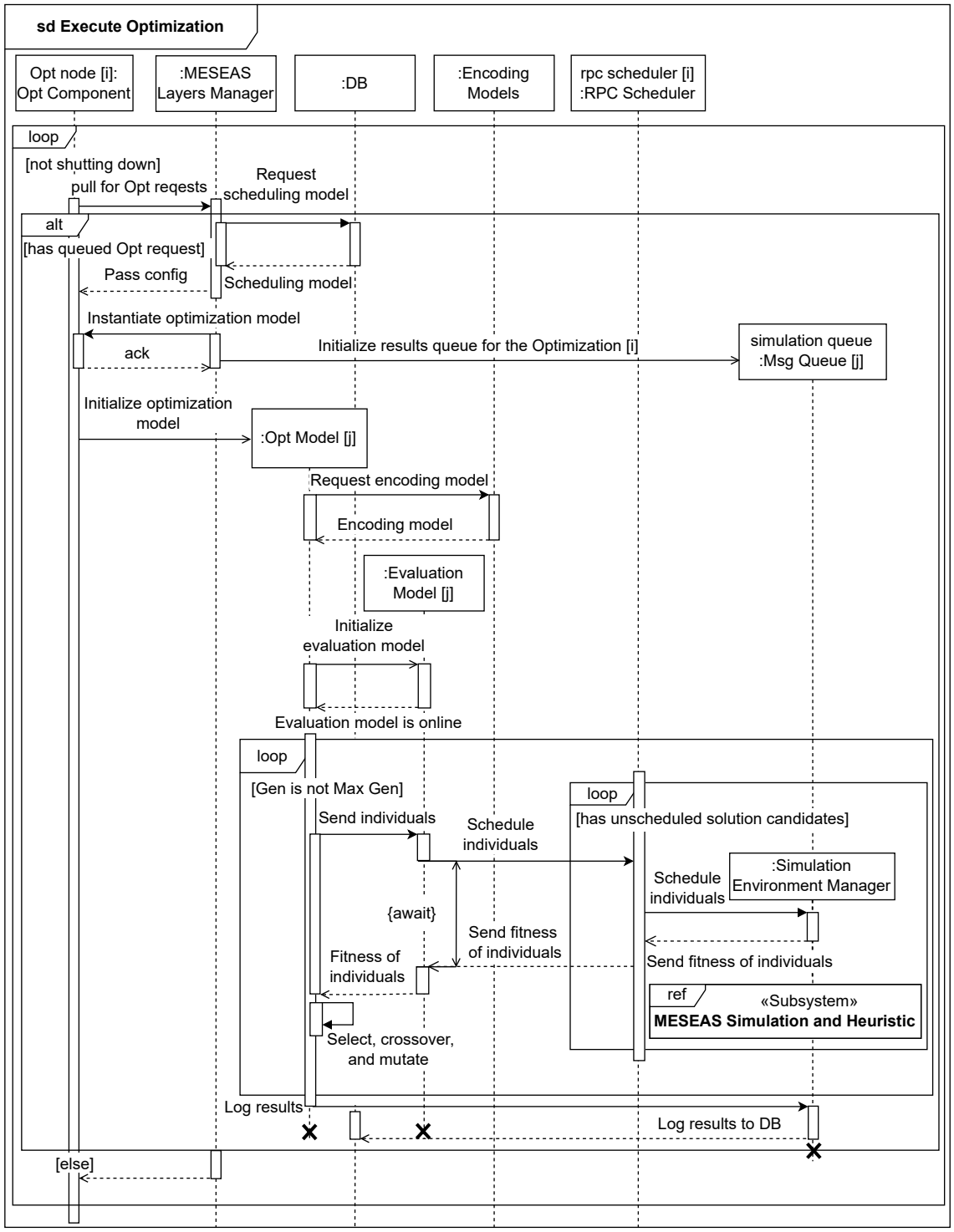


Figure 3.12: UML sequence diagram of the optimization component.

Upon receiving a request with the corresponding configurations, the layers manager retrieves the corresponding scheduling model from the database and passes it to the optimization component to initialize an optimization model. Based on the configurations, the necessary encoding model is retrieved to initialize the requested improvement algorithm with the requested encoding and start the client execution module. Then, the optimization model signals the evaluation to initialize and prepare the server execution module to be exposed to the optimization model.

Finally, the optimization process starts in the inner loop as depicted in Figure 3.12. The requested evolutionary algorithm starts with randomly generated solution individuals to form the first population. Afterward, the solution individuals of a generation are passed to the simulation environment manager, which distributes the evaluation of solution individuals among available simulation instances in the MEASEAS simulation and heuristic subsystem. Once all solutions individuals of a generation are evaluated and assigned fitness values based on the objective function, the results are pushed back to the optimization instance to execute evolutionary operators. After applying selection, crossover, and mutation functions, the next generation of solution individuals is ready to be evaluated by the simulation and heuristic subsystem.

This process is repeated until the maximum number of generations is reached to break the optimization process. Here, one could rely, for instance, on some convergence functions to break the optimization process automatically. The developed encoding models significantly influence the overall optimization process and define the shape of a solution individual (Osaba et al., 2021, p. 8; Talbi, 2009, p. 76). In the presented methodology, three encoding models are proposed and evaluated to solve scheduling problems.

3.6.3 DRL scheduling and evaluation models

To leverage the potential of DRL techniques for solving scheduling problems, we have to formulate a DRL scheduling model. Since most of DRL applications are evaluated in game-like environments, the objective is to transform the scheduling problem and develop an environment in which the DRL technique can be trained. DRL methods build up on the Markov Decision Process (MDP) to formalize a problem (Papadimitriou and Tsitsiklis, 1987a). An MDP model can be expressed using a tuple $\langle \mathcal{S} \mid \mathcal{A} \mid \mathcal{T} \mid \mathcal{R} \rangle$. Where $\mathcal{S}$ denotes the state of the environment that is perceived and observed by a DRL agent. It is often referred to as observation space, which is exposed to a DRL agent. $\mathcal{A}$ represents the shape of the action space that regulates how a DRL agent interacts with its surrounding environment. At some state of the environment $s \in \mathcal{S}$, the action taken by the agent $a \in \mathcal{A}$ is translated by a transition function $\mathcal{T}$ such that $\mathcal{T}(s_t, a_t, {s'}_{t+1})$. After taking the action and applying the transition function $s \xrightarrow{a} s'$, the agent receives the current observation of the state s'. Based on the observation space and taken action, the agent gets some reward r, which is computed using the reward function $\mathcal{R}$. Based on the MDP notations and the presented data model for scheduling, the action space, observation space, transition function, and reward function will be formalized in the coming subsections.

The state and observation space $\langle \mathcal{S} = (PI \cup X^* \in \mathbb{X}) \mid \mathcal{A} \mid \mathcal{T} \mid \mathcal{R} \rangle$: The observation space of a DRL agent is defined based on the shape of the problem instance PI, and the solution of the scheduling problem $X \in \mathbb{X}$ as detailed in Section 3.4.2 and Section 3.4.2, respectively. The problem instance and solution are obviously formalized to address scheduling concerns of the considered scheduling environment SP (cf. Subsection 3.4.1). After initializing a DRL agent using the presented methodology, the agent is exposed to the problem instances before starting the training. Then, after every action $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$, the constructed solution $X^* \in \mathbb{X}$ is passed to the agent based on the defined space of observation. The observation space following the MDP notation is expressed as $\mathcal{S} = (PI \cup X^* \in \mathbb{X})$.

Action space based on allocation heuristics $\langle \mathcal{S} \mid \mathcal{A} = HL\ (\Phi^A_\varphi) \mid \mathcal{T} \mid \mathcal{R} \rangle$: As discussed earlier in the formulation of the encoding models for optimization, scheduling concerns are addressed given some scheduling period $T = \{T_t, \ldots, T_{|T|}\} : \forall\ t \in \{1, \ldots, |T|\}$. To integrate the functionality of the optimization component with the machine learning component, the definition of the action space is crucial. The optimization component is developed using evolutionary algorithms and relies on the developed encoding models to search for high-quality solutions for solving scheduling problems. To utilize the obtained historical solutions from the optimization component for training a DRL agent, a compatible action space must be developed. Therefore, we rely on the discussed encoding models for solving scheduling problems, which we discussed in Subsection 3.6.2.

The action space of the adopted DRL methods is defined using constructive allocation heuristic markers. The HL component of the presented methodology encompasses the logic of a set of allocation constructive heuristic $A = \{A_a, \ldots, A_{|A|}\} : \forall\ a \in \{1, \ldots, |A|\}$. Given a selected set of allocation heuristics in HL, a DRL agent decides which allocation algorithm should be used at every decision point $T_\varphi \in T$ during a scheduling period. It implies that the number of possible actions at a decision point $T_\varphi \in T$ depends on the number of selected allocation heuristics. An agent takes an action represented by a discrete integer value Φ^A_φ which is pounded by the selected set of allocation heuristics such that $\forall\ \Phi_\varphi \in \{1, \ldots, |A|\}$ (cf. Subsection 3.6.1 and Equation 3.6).

In conclusion, if we map the definition of the action space in MEASES to the MDP notation, it can be expressed as $\boldsymbol{\mathcal{A}} = HL\ (\Phi^A_\varphi)$. Every action instantiates some allocation heuristic from the selected set A, which must be used to allocate jobs to machines at decision point T_φ. This formulation of the action space is sufficient if scheduling concerns in the considered environment do not require sequencing. Otherwise, an extended encoding that includes sequencing heuristics is discussed in the next subsection.

Action space based on allocation and sequencing $\langle \mathcal{S} \mid \mathcal{A} = HL\ (\Phi^A_\varphi,\ \Phi^B_\varphi) \mid \mathcal{T} \mid \mathcal{R} \rangle$: Suppose the operations of the considered environment are subject to sequencing. In that case, we may either rely on a fixed sequencing algorithm or develop an extended encoding of the action space, which facilitates the utilization of sequencing algorithms. This encoding follows the same logical design as the second optimization encoding model. The objective is to sustain a compatible shape of the action space, in case we want to rely on both com-

ponents to address scheduling concerns in a considered environment. However, enlarging the shape of the action space increases the complexity of the problem, which a DRL agent will be trained to solve.

We previously defined the set of available sequencing heuristic in the HL component of the MESEAS by $B = \{B_b, \ldots, B_{|B|}\} : \forall\, b \in \{1, \ldots, |B|\}$. To extend the previous definition of the action space, a DRL agent must be exposed to two controllers, A and B, to interact with the environment. At a decision point $T_\varphi \in T$, a DRL agent dictates which allocation algorithm $A_a \in A$ and which sequencing algorithm $B_b \in B$ must be used to schedule jobs on machines. Two discrete integer values represent the action $\boldsymbol{a} = (\Phi^A_\varphi,\ \Phi^B_\varphi)$ that are decoded by the HL to instantiate and utilize the corresponding allocation and sequencing heuristics, A_a and B_b respectively. The action space definition using this encoding can be summarized by mapping it to the MDP tuple notation such that: $\mathcal{A} = HL\ (\Phi^A_\varphi,\ \Phi^B_\varphi)$. According to the MDP model, the agent's actions must be evaluated using the transition function $\mathcal{T}$, which we will discuss in the coming subsection.

Transition function and the DRL environment $\langle \mathcal{S} \mid \mathcal{A} \mid \mathcal{T} = Sim\,(A_a,\ B_b,\ T_\varphi) \mid \mathcal{R} \rangle$: A DRL agent depends on a transition function, which translates and applies its actions in the corresponding scheduling environment. The presented methodology relies on simulation techniques to provide a transition function and a training environment for a DRL agent. Our objective was to integrate all functionality layers of the presented methodology and combine the utilization of simulation, heuristic, and improvement methods for addressing scheduling concerns. To start training a DRL agent using MESEAS, the simulation component instantiates a simulation model. This simulation model is a digital representation of a considered scheduling environment. It relies on the definition of the SP (cf. Subsection 3.4.1).

Based on the defined action space and observation space, a DRL agent may take an arbitrary action $HL\ (\Phi^A_\varphi,\ \Phi^B_\varphi)$ at T_φ. The actions of the agent dictate which allocation and sequencing algorithm may be used to deal with scheduling concerns. The simulation model of the considered environment decodes the received actions. After decoding the actions at time T_φ, the obtained A_a and B_b are employed during the simulation to allocate and sequence jobs to machines. The training of a DRL agent is an iterative process. Therefore, this process is repeated until some breaking criterion is met. After sufficient training, a DRL agent may drive an appropriate scheduling policy π_θ that can be used to deal with scheduling concerns. The quality of the achieved scheduling policy depends on the reward value, which the agent seeks to maximize. This reward value is computed based on a well-defined reward function, which we will elaborate on in the coming subsection.

Reward function $\langle \mathcal{S} \mid \mathcal{A} \mid \mathcal{T} \mid \mathcal{R} = -\Gamma_\omega \rangle$: Finally, the last element in the MDP tuple notation is the reward function. The reward function is used to estimate the quality of actions taken by a DRL agent during the training. A DRL agent is aware of its possible actions based on a well-defined action space $\mathcal{A}$. The transition function $\mathcal{T}$ is used to map the actions of the agent to its corresponding environment. Based on a

scheduling environment SP, a simulation model is built to prepare an environment for a DRL agent. This transition function generates an allocation and a sequencing map of jobs to machines, which is subject to some selected objective measures $\Gamma = \{\gamma_i, \ldots, \gamma_{|\Gamma|}\} : \forall\ i \in \{1, \ldots, |\Gamma|\}$. At each step $\boldsymbol{s} \in \mathcal{S}$, the agent seeks to maximize its reward subject to the reward function. Therefore, the reward function must be defined as a maximization function of the normalized objective values. Thus, we can rely on the weighted sum objective function discussed in Section 3.4.2 to formulate the reward function $\mathcal{R} = -\Gamma_\omega$ as depicted in Equation 3.9.

$$\arg\max_{X \in \mathbb{X}} \Gamma(X) = \arg\max_{X \in \mathbb{X}} \left(-\sum_{i=1}^{|\Gamma|} \omega_i \cdot \gamma_i(X) \right) \tag{3.9}$$

Subsection 3.4.3 detailed the definition of the presented methodology for solving scheduling problems. Given the definition $MESEAS_{(X,\ T)} = \langle SP \mid PI \mid Sim \mid HL \mid \boldsymbol{ML} \mid \Gamma \rangle$, we may summarize and map the elements in this definition to the MDP model as presented in Equation 3.10

$$\langle \mathcal{S} = (PI \cup X^* \in \mathbb{X}) \mid \mathcal{A} = HL\ (\Phi^A_\varphi,\ \Phi^B_\varphi) \mid \mathcal{T} = Sim\ (A_a,\ B_b,\ T_\varphi) \mid \mathcal{R} = -\Gamma_\omega \rangle \tag{3.10}$$

3.6.4 Design of the machine learning component

The machine learning component of the MESEAS methodology comprises two main subcomponents, namely, a DRL model and a DRL environment. The design and architecture of these two subcomponents rely on the previously mapped MDP model to the core components of the presented method (cf. Equation 3.10). Figure 3.13 depicts the main logical modules of the ML component using a UML component diagram.

The modularity in the design of the component ensures further development and maintenance of adopted DRL algorithms. It also facilitates the adoption and integration of further state-of-the-art DRL methods without restructuring the entire architecture. The component's architecture is developed and implemented using client-server computing architecture to facilitate the distribution of workload generated by training a DRL agent and evaluation of actions between available physical computing resources. As illustrated in Figure 3.13, the ML component is also managed by the MESEAS layers manager, which routes requests and fetches required data for the initialization and execution of the machine learning subcomponents.

The DRL environment subcomponent encompasses the DRL scheduling model, the evaluation model initializer, and an environment server. The DRL scheduling model consumes supplemented configuration from the layers managers and initializes the shape of the observation space for a given scheduling problem based on the received data. As depicted in Figure 3.13, the evaluation model initializer pre-processes the retrieved encoding model and the objective values of the received configuration. Then, it designates the step of a DRL agent, the rest method to start a new training episode, the reward function used

to evaluate the actions of an agent, and the action space used by the agent to interact with the environment. After initialization is completed, the evaluation model initializer triggers the DRL model subcomponent and executes the environment server to start awaiting actions from the DRL Model subcomponent.

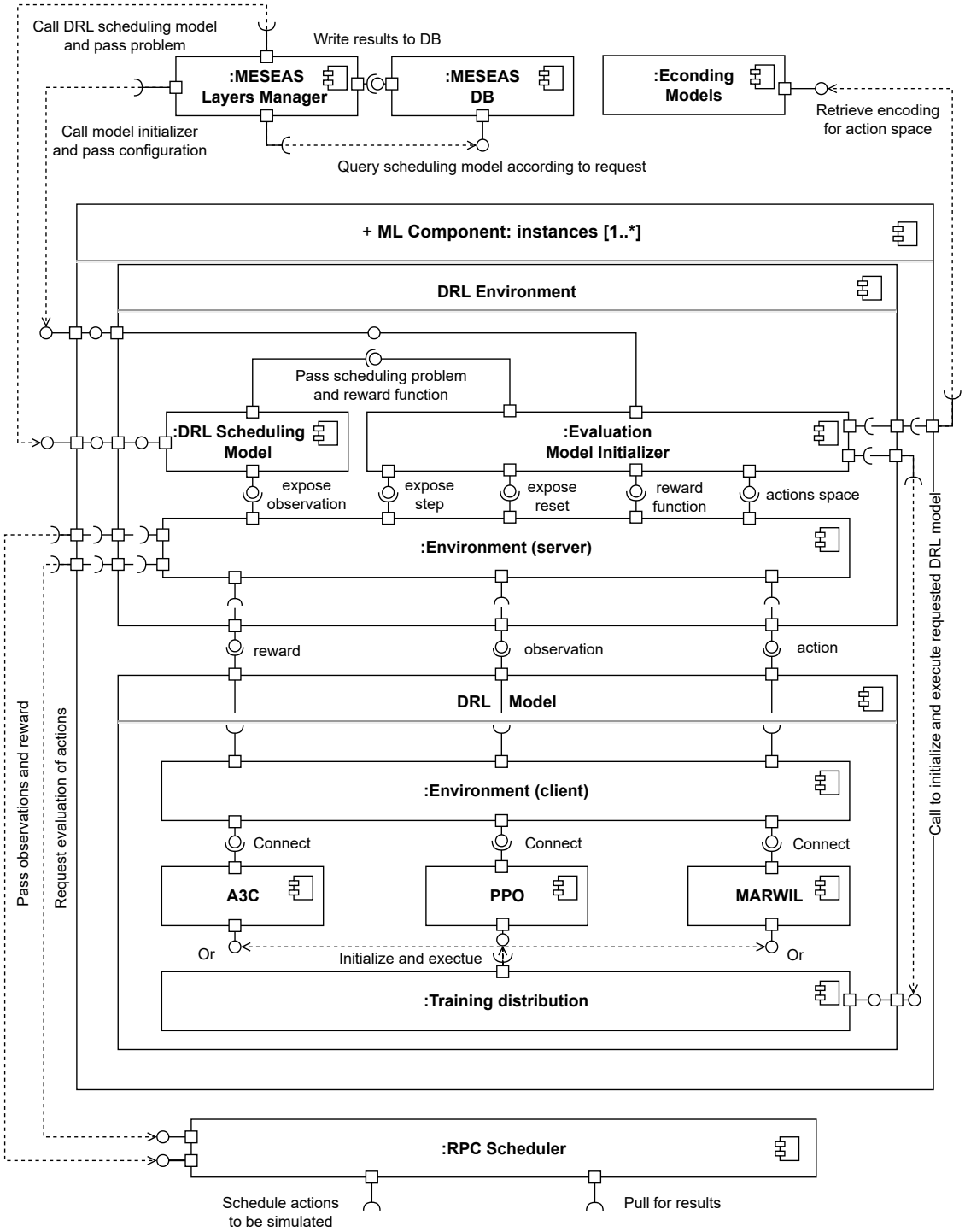


Figure 3.13: UML Component diagram of the DRL component.

The DRL environment further acts as a logical interface between the DRL model and the simulation and heuristic subsystem of MESEAS methodology. The DRL model sub-component consists of several logical modules: The training distribution module, the adopted Asynchronous Advantage Actor-Critic (A3C) execution module, the adopted Proximal Policy Optimization (PPO) execution module, the adopted Monotonic Advantage Re-Weighted Imitation Learning (MARWIL) execution module, and the environment client module (cf. Figure 3.13). The DRL model is initialized and executed based on the configuration passed on by the layers manager and the DRL environment subcomponent. Based on the configurations received by the layers manager, the DRL environment subcomponent triggers the initialization of a trainer instance/s of the desired algorithm. We rely on a parallelization framework that instantiates multiple instances of the selected agent and handles merging training results. Some implementation details will be discussed in Chapter 4. Finally, an environment client is executed to start the training process. The environment client relies on the exposed interface by the DRL environment to send actions and receive observations and rewards. The DRL environment subcomponent pre-processes and synchronizes the flow of actions, observations, and rewards between the DRL model, the simulation, and the heuristic subsystem through an RPC scheduler.

Figure 3.14 depicts the execution of the machine learning component using a UML sequence diagram. The sequence diagram highlights the interactions, the lifelines, and the exchange of data between the central logical modules of the machine learning component. Based on the configuration, the machine learning instance is online, awaiting to start training agent/s. Upon receiving a request from MESEAS layers manager with the required configurations, the ML component initializes a DRL environment. The DRL environment subcomponent retrieves the requested encoding model to construct a DRL scheduling model and an evaluation model. Then, a DRL model is initialized and signals its readiness to start training. The ML component of MESEAS methodology is also designed to be used by users with different levels of expertise in the field of DRL.

Finally, the DRL training is initiated as illustrated in the loop in Figure 3.14. Actions of the DRL agents are generated by the DRL model subcomponents and passed to the DRL environment server. The DRL environment component schedules received actions for evaluation by MESEAS simulation and heuristic subsystem through the RPC scheduler. The actions are received by the simulation environment managers and processed by the available simulation instances of this DRL environment. Upon completing the evaluation of actions, the corresponding observations and rewards are passed to the DRL environment component. The final results are processed and supplied to the DRL model subcomponent. This rationale is repeated until the training is completed. Usually, the breaking creation to stop training an agent can be defined based on the number of training steps or based on the quality of achieved solutions.

The formulation of the DRL scheduling model plays a major role in successfully training a DRL agent. DRL techniques are often evaluated and adopted to interact with game-like environments (Kanervisto et al., 2020, p. 479). Therefore, we developed various DRL encoding models with different levels of complexity to address scheduling problems.

The DRL problem encoding must be compatible with the problem encoding used in the optimization component. The compatibility facilitates combining the utilization of DRL methods with evolutionary methods to address scheduling concerns. The developed DRL encoding models expose the agent to a scheduling problem, like a game in which the agent interacts with a simulation instance and selects allocation and/or sequencing heuristics for scheduling during a scheduling period.

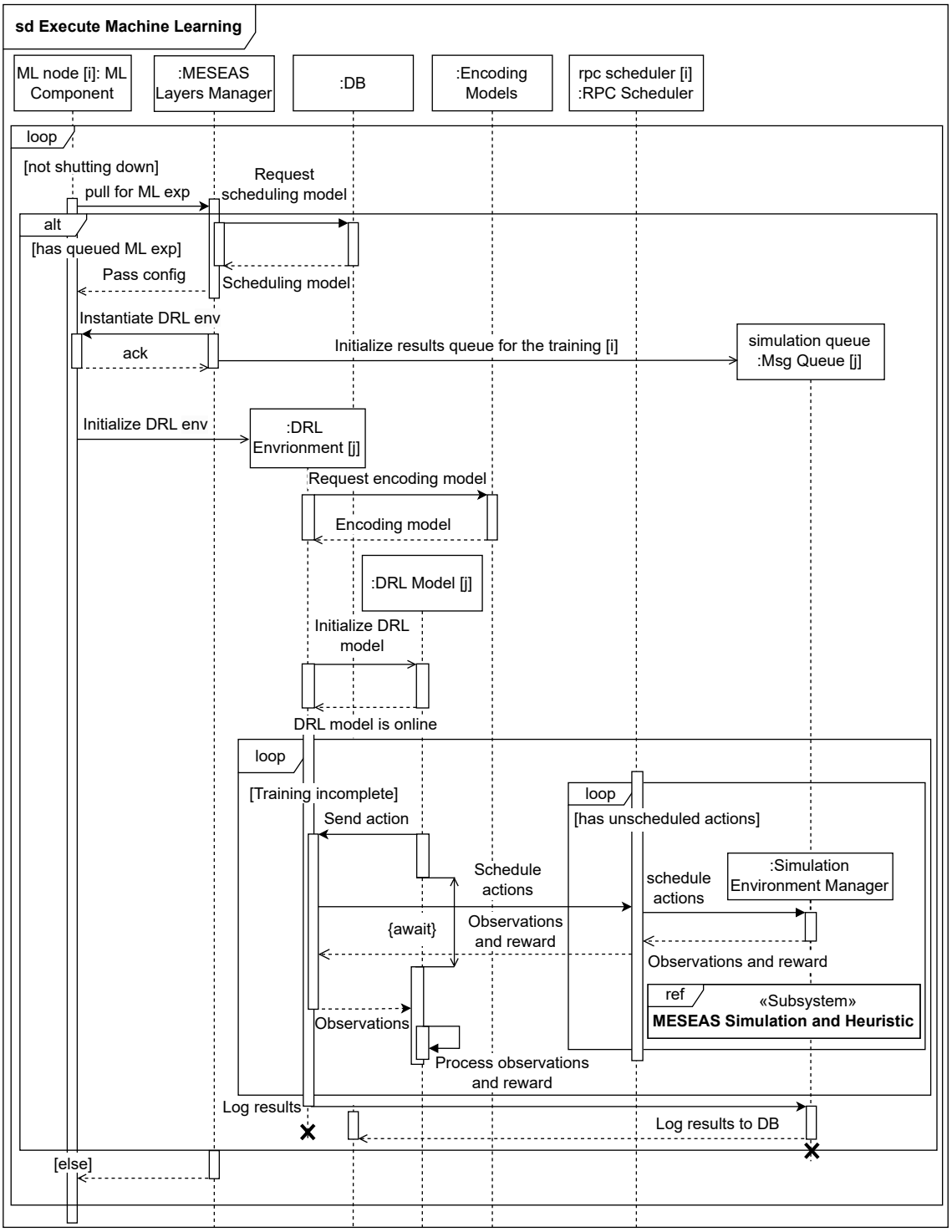


Figure 3.14: UML sequence diagram of the DRL component.

3.6.5 Design evaluation of optimization and machine learning components

The designs of the optimization component and machine learning components are evaluated during the prototyping and implementation stage of MESEAS methodology. Our initial investigations of scheduling problems show that constructive heuristic techniques lose performance when the scheduling problems are solved considering multiple objective values (Nahhas et al., 2017a; Nahhas et al., 2018a). The presented results in both research largely agree with the findings presented by (Varasteh and Goudarzi, 2017).

In (Nahhas et al., 2018b), we explored combining the adoption of heuristic and improvement methods for solving scheduling problems in the manufacturing environment $\langle HFS_4 \ (P_4, P_5, 1, 1) \mid f_{g,h}, brkdwn \mid C_{max}, \sum T_j \rangle$ and investigated the impact of constraints on the computational effort required to solve scheduling problems. Shortly after, we proposed the first prototype of the artifact to address scheduling problems in cloud environments in (Nahhas et al., 2019a). In this research, we systematically analyzed switching between different constructive heuristics during a scheduling period to achieve better performance. The algorithm selection at each switching point was made using a control strategy based on GA. The scheduling problem was considered to be subject to the minimization of energy consumption while considering performance constraints $\langle P_m \mid D_{pr}, M^C \mid E, U \rangle$. The experimental results showed that utilizing different heuristics during a scheduling period minimizes energy consumption while considering performance constraints compared to the individual heuristics. We presented extended evaluations of the design and the rationale of the optimization component in (Nahhas et al., 2021a) and (Nahhas et al., 2021b). The former study investigated addressing large-scale single-stage scheduling problems in cloud environments $\langle Q_m \mid M^C \mid E, \ U \rangle$. The problems are solved to minimize the overall energy consumption while considering performance concerns. The later study presented an extensive evaluation of combining constructive and improvement methods with simulation techniques for solving scheduling problems with multiple objective values $\langle HFS_4 \ (Q_5, \ Q_5, \ Q_5, \ P_2) \mid f_{g,h} \mid C_{max}, \ MS, \ T, \ U \rangle$. Detailed insights into the computational results will be discussed in the next chapter (cf. Chapter 4).

The design of the machine learning component followed up after the optimization component and the new implementation of the artifact using open-source technologies. In (Nahhas et al., 2022b), the first prototypical implementation of the machine learning component is presented. In this research, we aimed to investigate the adoption of DRL methods for solving scheduling problems subject to multiple objective concerns. We developed a DRL scheduling model and adopted PPO, resented originally in (Schulman et al., 2017b) and A3C, which is proposed by (Mnih et al., 2016). The design of the machine learning component, which is based on DRL, is evaluated for solving multi-stage scheduling problems in manufacturing $\langle HFS_4 \ (Q_5, \ Q_5, \ Q_5, \ P_2) \mid f_{g,h} \mid C_{max}, \ MS, \ T, \ U \rangle$. The DRL agent interacts with an environment, which is provided by the simulation component, and selects allocation and sequencing heuristics to be used during the scheduling period. The computational results show that the agents learn to solve the mentioned scheduling problems given the formulated DRL scheduling model, resulting in a

successful scheduling policy. The performance of the ML component is compared to related works presented by Lang et al. (2021b) for solving a two-stage scheduling problem $\langle HFS_2\ (P_5,\ P_4) \mid f_{g,h} \mid C_{max},\ MS,\ T \rangle$. The presented research was awarded as the best paper award by the international scientific committee of the conference in the analytic and decision science track. Further insights into the results in terms of objective values will be thoroughly discussed in Chapter 4.

3.7 Summary of MESEAS design

The problem statement and thesis objective of the presented artifact were derived based on the analysis presented in the first and second chapters of the thesis. This chapter presented and discussed extensively the design of the artifact before implementation to achieve the research objective of the thesis. First, the function and nonfunctional requirements were detailed and grouped to construct the overall functional structure of the artifact. The initial conceptual design of MESEAS to integrate simulation, heuristic, improvement, and machine learning methods was presented using a UML information flow diagram. This conceptual design was later detailed from an engineering perspective using UML component diagrams to present the components grouped into subsystems of MESEAS. Subsequently, the designs of components in MESEAS were thoroughly discussed and elaborated using mathematical models, pseudocode, UML component diagrams, and UML sequence diagrams. Throughout the design phase of MESEAS components, preliminary evaluations were conducted to verify intermediate designs and identify possible challenges. Hence, the modeling and simulation components section, the heuristic library components section, and the optimization and machine learning components sections were concluded with design evaluations.

4 Evaluation

The work at hand relies on the Design Science Research (DSR) framework presented by von Hevner et al. (2004) to conduct the research systematically and conclude the final artifact. von Hevner et al. (2004, p. 78) DSR framework relies on the suggested main design activities by March and Smith (1995, p. 255) to produce IT artifacts. Later on, this practice was well-established by the DSR research community and regarded as an evaluation pattern *(Build-Evaluate)*. This evaluation pattern was later detailed with evaluation activities, input, output, exemplary evaluation criteria, and exemplary evaluation methods in the work presented by Sonnenberg and vom Brocke (2012, p. 393). This work adopts the DSR framework by von Hevner et al. (2004) and relies on the detailed evaluation pattern presented by (March and Smith, 1995). Table 4.1 displays the evaluation activities suggested by Sonnenberg and vom Brocke (2012) with its corresponding input and presumed output at each evaluation activity.

Table 4.1: DSR evaluation activities adapted based on Sonnenberg and vom Brocke (2012)

Activity	Input	Output
Eval 1	Problem statement, research need, design objectives, design theory, and existing solution.	Justified problem statement, research gap, design, and objectives.
Eval 2	Design specification, design objectives, design tool, and design methodology.	Validated design specification, justified design tool/ methodology
Eval 3	Instance of an artifact (prototype)	Validated artifact instance in an artificial setting (proof of applicability)
Eval 4	An instance of an artifact	Validated artifact instance in a naturalistic setting (proof of usefulness)

The first evaluation activity (*Eval 1*) requires a clearly defined problem statement or an observation of a problem, which necessitates research and development effort to be addressed. This problem must be associated with a design objective and theory. The presumed outcomes of this evaluation activity include a justified problem statement, a research gap, a design, and an objective. The first evaluation activity will be summarized in Section 4.1. The second evaluation activity (*Eval 2*) necessitates a specification of the designed artifact in the form of functional and non-functional requirements. These design

specifications must be associated with design objectives that are incorporated in the body of the proposed research artifact. The design specifications of the artifact are developed using clearly stated design tools and a methodology. The results of this evaluation activity demonstrate a validated design specification that is based on validated design tools and methodology. The second evaluation activity will be concluded in Section 4.2.

The third evaluation activity (*Eval 3*) conditions an instance of the artifact during development to ensure that the design specifications can be realized and the artifact can be applied (*proof of applicability*). The fourth evaluation activity (*Eval 4*) conditions an instance of the developed artifact to ensure that the design specifications are realized and the artifact can be applied in the respected environment to address business concerns (*proof of usefulness*). The third evaluation activity spans multiple sections to demonstrate the instantiation and utilization of the presented methodology for addressing scheduling concerns in the cloud and manufacturing environments. The fourth evaluation activity is partially covered in the manufacturing application field.

The detailed evaluation pattern of Sonnenberg and vom Brocke (2012) largely builds up on the DSR process model presented originally by (Peffers et al., 2007, p. 93). The presented evaluation activities are also associated with certain evaluation criteria and methods, which can be adopted during every step of the research and development processes of the artifact (Sonnenberg and vom Brocke, 2012). Sonnenberg and vom Brocke (2012) suggested the evaluation criteria for DSR artifacts based on analysis of related works and evaluation patterns that rely on the work presented by (March and Smith, 1995, p 255). Sonnenberg and vom Brocke (2012) emphasized the nature of the evaluation setup, which is applied to evaluate an artifact.

Here, the authors defined the nature of the evaluation activity following the concept of the three realities originally discussed by Sun and Kantor (2006, p. 616). The three realities to which a DSR research artifact is evaluated subject to are: "real system", "real problems", and "real users" Sun and Kantor (2006, p. 616). The concept detailed an evaluation setup of a proposed research artifact in the IS discipline. The authors here aimed to stress the naturalistic evaluation of an artifact (Venable, 2006, p. 186). Table 4.2 details the evaluation steps and activities that are conducted to evaluate the presented methodology. Here, we want to stress the partial nature of the fourth evaluation activity.

Based on the adopted evaluation pattern by (Sonnenberg and vom Brocke, 2012) and the concept of three realities, the fourth evaluation step does not meet the full integration of the artifact in an organizational context „... artifact instances that are fully embedded within the organizational context.“ (Sonnenberg and vom Brocke, 2012, p. 396). Despite fulfilling the requirements of three realities, a proper evaluation in the context of the fourth activity would require full integration of the artifact in an organization. Then, systematically monitoring its impact on the daily operation of the organization would conclude the fourth evaluation activity. The full integration of the artifact and monitoring of its performance are future research and development activities. Hence, this constitutes a limitation in the evaluation of the presented artifact.

Table 4.2: Applied evaluation activities in this work

Activity	Evaluation criteria	Evaluation methods
Eval 1	Applicability, suitability, importance, and novelty.	Literature analysis and descriptive investigation of existing methods.
Eval 2	Feasibility, accessibility, understandability, clarity, and consistency.	Logical reasoning, descriptive analysis, demonstration, and simulation.
Eval 3	Suitability, solution quality, effectiveness, and efficiency.	Demonstration with a prototype, and experiment with a prototype.
Eval 4 - (partial)	Applicability, efficiency, and fidelity with real-world phenomenon.	Case study

4.1 Summary of the first evaluation activity (Eval 1)

The first evaluation stage starts with a problem statement or consistent observation of the problem that originated from the application environment. The problem statement must reflect the business needs to derive appropriate requirement specifications in the later stages. The problem statement is introduced and elaborated in Chapter 1. From a research perspective, most scheduling problems are complex and NP-hard combinatorial optimization problems. From a practical perspective, scheduling activities are daily decision-making tasks in the normal operation of cloud, manufacturing, or service environments. These two perspectives are thoroughly discussed throughout the first and second chapters of the presented work (cf. Chapter 1 and Chapter 2). The main objective of their extensive discussion and analysis is to demonstrate the *importance* and the *relevance* of scheduling problems. Establishing the importance and relevance of the problem is the first step in pursuing proper solution methods.

In the second chapter, we dedicated Section 2.3 to discuss the prominent conventional solution methods of scheduling problems and highlight their potential and limitations. Conventional solution methods are either computationally too expensive or lack the quality of the solution for solving complex scheduling problems. Thus, their *suitability* and often *applicability* in addressing complex scheduling concerns are limited. Therefore, we seek a solution methodology that integrates the utilization of simulation, heuristic, improvement, and machine learning methods for addressing scheduling concerns. To achieve this research objective, we explored and studied the relevant foundations of the knowledge base to infer proper design and rely on the corresponding theories.

Therefore, we conducted a descriptive investigation of prominent methods to analyze their performance in solving scheduling problems (cf. Section 1.4) and validate the initial observations. The obtained descriptive overview of the existing methods was simultaneously compared against the findings of the literature analysis. The concluded observations

in Subsection 2.3.3 are consistent with many recent research endeavors, which justify the problem statement and highlight the research gap. As a result, research and development activities to realize the intended design and achieve the stated objectives are required. The results of these analyses are published to receive and incorporate feedback from the research communities. Based on the problem statement, research objective, and the investigation of conventional solution methods, a final analysis of the literature is conducted to identify the most significant works related to the proposed design. Section 2.5 demonstrated the adopted methodology and results of the literature review. The section discussed identified related works and established the *novelty* of the intended design.

4.2 Summary of the second evaluation activity (Eval 2)

Based on the problem statement, the research objective of the thesis is formulated. To accomplish the stated objective in DSR, research questions must be answered using proper *design methodology* (cf. Subsection 2.5.1). Based on the formulated research questions, the design specifications of the intended artifact are formulated. These specifications are profoundly discussed by means of functional and non-functional requirements, which the intended artifact must fulfill (cf. Subsection 3.1.1 and Subsection 3.1.2). To associate the stated *design specifications* with the research objective, we grouped them into functionality layers that define the overall functional structure of the intended methodology and dictate *the design objective* of the thesis (cf. Subsection 3.1.3).

The intended artifact was developed with modularity in mind to ensure that functional components are logically independent and easy to operate and maintain. To achieve the artifact's design objective, we utilized various design tools and methodologies to define the overall structure and detail its components. First, we relied on Unified Modelling Language (UML) as a *design tool* to sketch the initial blueprints of the artifact using UML information flow diagrams and component diagrams (cf. Section 3.2 and Section 3.3, respectively). Based on the abstract representation and the initial blueprint of the artifact, we utilize mathematical modeling a *design tool* to develop the logical design and specify the rationale of the artifact. Mathematical modeling is well suited to describe the necessary data structures for integrating methodologically different techniques such as simulation, heuristic method, evolutionary method, and DRL methods (cf. Section 3.4). Based on the defined data structures of the artifact, we relied on mathematical modeling to detail the rationale of the individual components of the artifact.

Based on the artifact's overall data structures and the overall design blueprint, we utilized UML component diagrams to infer the initial blueprints for the simulation, optimization, and machine learning components. The design of the heuristic library component is presented using mathematical modeling and pseudocode convention to ensure a sufficient level of detail and clarity. Finally, UML sequence diagrams are to explain the lifelines and information flow between various components during executions (cf. Section 3.4, Section 3.5, and Section 3.6).

The objective of *Eval 2* activity is to validate the proposed design specification of

the intended artifact and justify the design tools and methodology before starting the development phase. In summary, a mathematical modeling design tool is fundamental to formalizing the required data structures of the artifact design. We utilized UML design tools to present information flow, component, and sequence diagrams for the intended components of the artifact. That is to achieve *understandability*, *clarity*, and *consistency*. Meanwhile, pseudocode and mathematical modeling are suitable for formalizing data structures and the rationale of developed and adopted algorithms. *Eval 2* activity is also a bridge between the design and construction phases in the process of artifact's development (Sonnenberg and vom Brocke, 2012, p. 392). Therefore, during the design phase of the research artifact, we evaluated its components and investigated their integration. We relied on *simulation methods* for evaluation and compared the initial design of components against the existing conventional solution methods. We summarized the conducted evaluations in Subsection 3.4.5, Subsection 3.5.3, and Subsection 3.6.5. The collected evaluation results are published to communicate the findings with research communities and integrate further feedback. The conducted intermediate evaluation ensures the *feasibility*, *accessibility*, and *understandability* of the proposed approach.

4.3 Implementation and deployment overview of the artifact (Eval 3.1)

The final prototype of the presented methodology is implemented using open-source and cloud-native technologies. The overall objective is to facilitate the deployment of MESEAS methodology using in-house infrastructures, cloud infrastructures, or hybrid infrastructures, where cloud resources are scaled down if they are not needed. Figure 4.1 depicts an execution overview of the prototype. We developed the final prototype to support multi-architecture infrastructure. It can be deployed and hosted on heterogeneous hardware in terms of computer architecture.

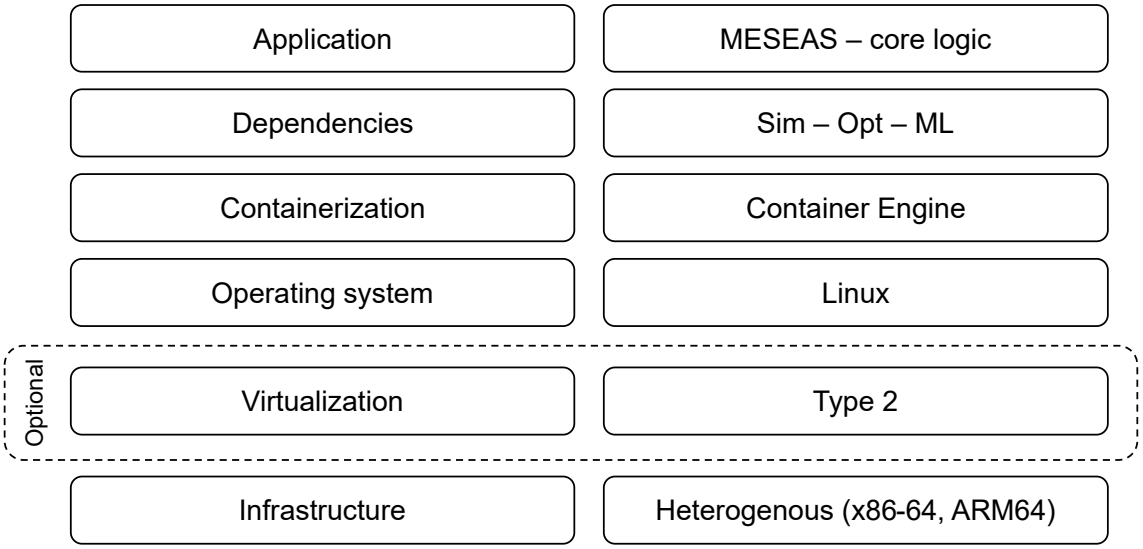


Figure 4.1: Execution overview of MESEAS methodology.

Depending on the hardware and the deployment strategy, virtualization is recommended but not mandatory to achieve scalability and full abstraction from the underlying infrastructure. On top of the virtualization, we relied on Linux operating systems to host and maintain the dependencies of MESEAS methodology. To achieve lightweight integration and full isolation from hardware, we adopted containerization techniques to incorporate the various independent modules of MESEAS subsystems. Containerization facilitates efficient execution, maintenance, and further development of various Simulation (Sim), Optimization (Opt), and Machine Learning (ML) dependencies and required technologies. Finally, the application level in the figure comprises the core logical modules of the MESEAS methodology, which we extensively discussed in the design chapter (cf. Chapter 3). They are managed independently from their underlying required technologies to achieve efficiency in operation and maintenance Subsection 4.3.2.

4.3.1 Multi-architecture deployment

We will briefly discuss the two most significant computer architectures and their differences to elaborate on the potential of combining them. IBM introduced the computer architecture of the IBM system/360, outlining distinct objectives to achieve general-purpose computers in the mid-sixtieth (Amdahl et al., 1964, p. 87). The main idea was to rely on standards in designing computer architecture so that computers of completely different hardware characteristics could execute the same software (Waterman, 2016, p. 1). That, in turn, established a new model in computer architecture that relies on the Instruction-Set-Architecture (ISA) as a distinct abstract interface that resides between the hardware and software. ISA dedicates how software interacts and utilizes a Central Processing Unit (CPU) of computer devices. The early published works, which laid out the groundwork for Instruction-Set-Architecture, were first presented in the late fortieth by (Burks et al., 1947). The authors' work was further summarized and presented in (Burks et al., 1982).

Currently, two general-purpose computer architectures are dominant, both of which rely on ISA: Complex-Instruction-Set Computers (CISC) and Reduced-Instruction-Set Computers (RISC) (Patterson, 2018, pp. 27-28). In the late seventieth, Intel introduced the first 8086 ISA processors with a 20-bit address size register, which was further developed to a 32-bit address size, leading to what is known as (80x86) or (x86) architecture. This architecture dominated the general-purpose computer world for at least 15 years (Patterson, 2018, p. 27). The x86 microprocessors belong to the family of CISC general-purpose computer architecture, which dominated the personal computer market for decades after introducing the 64-bit version by AMD (Patterson, 2018, p. 28). The RISC-based microprocessors still significantly dominate the market of embedded devices (Patterson, 2018, p. 27; Waterman, 2016, p. 11). Recently, the new 64-bit Advanced RISC Machine (ARM64) or ARMv8-based architecture has introduced highly efficient and power-conservative microprocessors. The popularity of ARM64 microprocessors is evident in reaching out to the personal computer and server computing industries. Hence, the presented artifact is developed with a heterogeneous multi-architecture hardware setup in

mind. MESEAS is deployed in this example on 16 physical nodes as depicted in Figure 4.2.

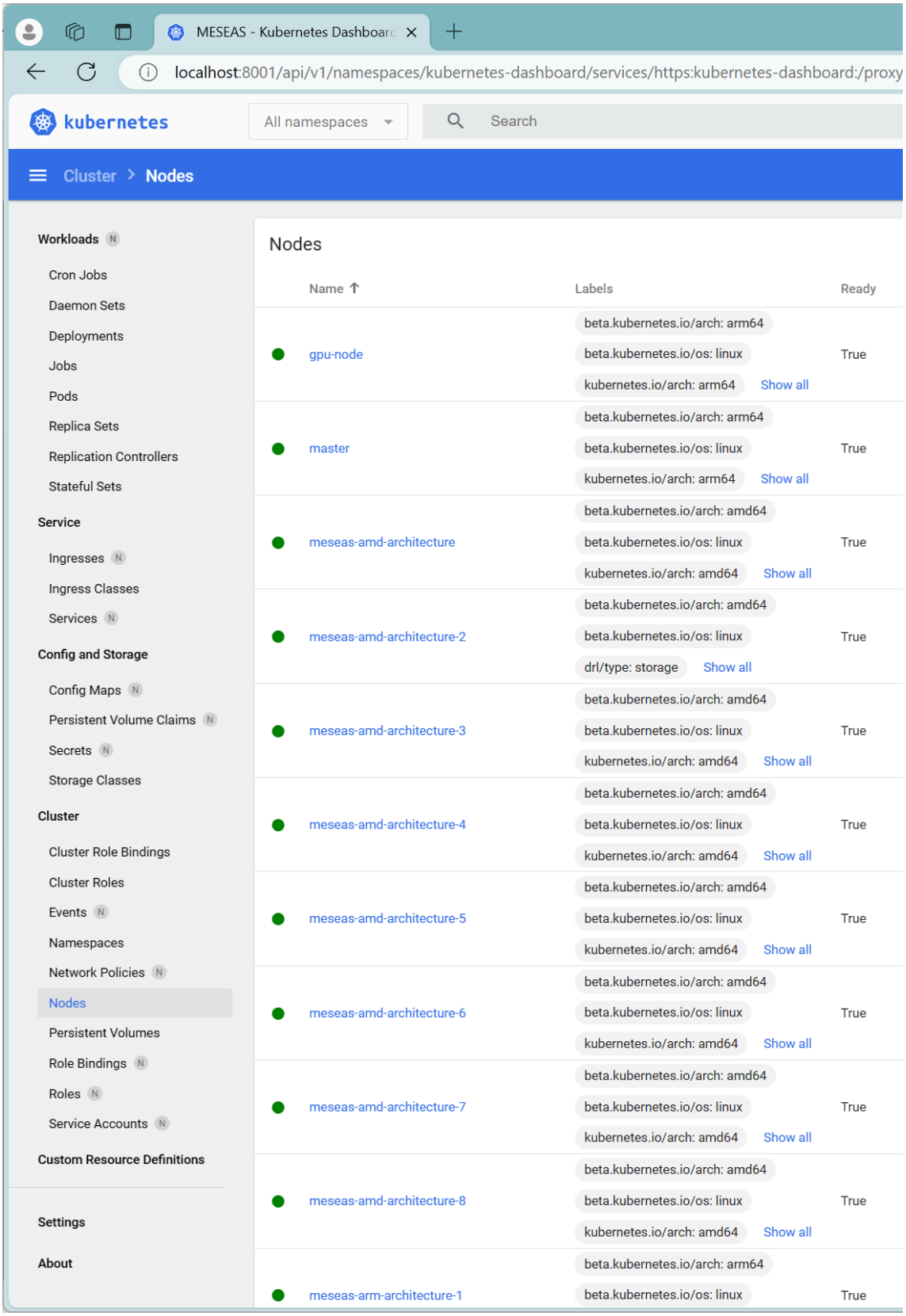


Figure 4.2: Multi-architecture deployment of MESEAS methodology using Kubernetes.

The physical setup consists of eight x86-64 physical nodes in addition to eight ARM64 physical nodes to leverage the potential of combining them. A multi-architecture deployment of the artifact facilitates flexible selection of the underlying infrastructure that is used to host and run the artifact. For instance, we may rely on x86-64 computer architecture hardware to host and run optimization and machine learning components of the presented artifact. Meanwhile, ARM64 computer architecture hardware is sufficient to host and run simulation instances of the artifact. Since AMD corporation[1] was the first to introduce the 64-bit version of ISA, *x86-64* or *amd64* have been common terms to refer to devices that rely on the 64-bit architecture.

The developed prototype of the artifact and its components is compiled for execution on both x86-64 and ARM64 hardware. Despite the evident outperformance of x86-64 hardware architecture in terms of computing power, utilizing ARM64 hardware reduces the overall power consumption required to host and use the artifact. The presented deployment relies on virtualization technologies to fully abstract available resources and facilitate proper scalability. Incorporating a virtualization layer remains operational and strongly depends on the hardware setup on which the artifact is deployed. In the prototypical final deployment, we relied on Type2-virtualization to integrate some physical nodes and installed the operating system directly without virtualization on other nodes. As demonstrated in Figure 4.2, we relied on Kubernetes to manage and maintain the orchestration of the artifact components.

4.3.2 Cluster implementation and deployment

Linux-based operating systems are the most suitable option for executing and maintaining the components of MESEAS methodology. They are ideal for running container engines and maintaining running applications efficiently. Linux operating systems provide various essential features such as namespaces, control groups for isolation of running containers, and other open-source technologies. For instance, Kubernetes can be easily integrated to manage infrastructure and workload. A container is an isolated software environment that encapsulates the source code and logical module of some application in addition to the required dependencies to execute the application (Boettiger and Eddelbuettel, 2017).

Containerization: The containerization of MESEAS is implemented using the Docker container engine. Docker[2] is an open-source technology for applying containerization strategies. It is often utilized to design, deploy, and manage software applications Engelmann et al. (2024). The popularity of Docker can be explained by its simple design. For instance, a Docker file and a Docker image are required fundamental container elements. A Docker file contains a set of commands that define the operating system's nature, required dependencies, and the source code of some applications (Merkel, 2014). This file is then executed to build a proper container image with the listed operating system, dependencies, and source code. It also facilitates a multi-stage building of container images, which we

[1] https://www.amd.com/en.html[Last checked on 17.01.2024]
[2] https://www.docker.com/[Last checked on 01.02.2024]

rely on to develop and maintain MESEAS individual components efficiently. A Docker image can be executed to construct and run a container that hosts the source code of the desired application. Containerization strategies enable quick deployment of the applications and facilitate distributed microservice architectures. It is ideal for facilitating flexible scaling of applications and services. The Docker platform supports Linux, Windows, and macOS operating systems. However, running on Windows requires a functional Linux subsystem. At the time of writing, the latest stable version of Docker Engine is 24.0.

Cluster deployment using Kubernetes: To manage seamlessly the deployment, operation, and automation of container management, we relied on Kubernetes technologies. Kubernetes is an open-source platform for the automated deployment, scalability, and management of containerized applications and services [3]. The core operational unit of Kubernetes is a pod. Figure 4.4, depicts an example physical node from the deployment of MESEAS. For instance, on this node, 16 containers (pods) are hosted and running. Following Kubernetes terminology, containers are organized into pods that are hosted on the Kubernetes nodes. Figure 4.3 demonstrate the deployment architecture of MESEAS methodology using Kubernetes.

The platform is based on a master-slave architecture, which comprises at least one master node and worker node/s. The master node hosts and maintains the so-called *"control plane"*. The control plan manages the communication with the worker nodes and ensures the efficient allocation of resources. It consists of an API-server, etcd, a kube-scheduler, and a controller-manager. The API server, *"kube-apiserver"*, handles the communication of the control plane, which is the most significant component in a Kubernetes cluster. The *"etcd"* is a distributed key-value store in the Kubernetes master node, which is primarily used to store the most important data of a Kubernetes cluster. It stores and maintains configuration data, running workload states, and metadata. Workload data of Kubernetes cluster include the states and health of pods, services, or replication controllers. The *"kube-scheduler"* is responsible for scheduling unallocated pods to available physical nodes that can meet its requested hardware requirements. Finally, the controller-manager runs cluster control processes, which supervise and react to the state of worker nodes, jobs, or service accounts. For instance, a node-controller process watches over the health of worker nodes in a Kubernetes cluster and reacts when they go down. The master node is the most fundamental component of the orchestration system in Kubernetes.

To successfully join a worker node to a Kubernetes cluster, three Kubernetes components must be installed and configured appropriately: a kubelet, a kube-proxy, and a container runtime interface. The *"kubelet"* is a service that runs on every Kubernetes node. It registers the node using apiserver and executes pod specifications from the master node. It essentially manages containers that are spawned by the cluster. The *"kube-proxy"* is the second fundamental service on every node, which acts as a local control plane on the worker nodes. It connects a node with the master node and maintains communication with the underlying network interface on the node (Kaiser et al., 2022).

[3] https://kubernetes.io/[Last checked on 21.03.2024]

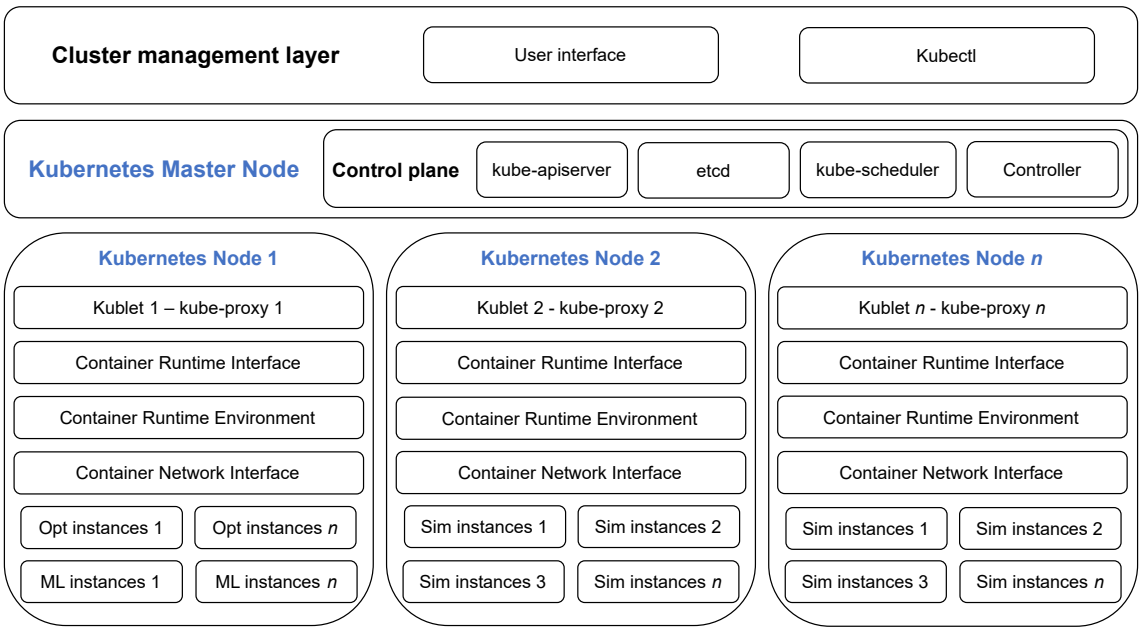


Figure 4.3: Multi-architecture deployment of MESEAS methodology: Technology stack.

Finally, a *"Container Runtime Interface (CRI)"* is essential for the kubelet to communicate with the underlying installed container runtime environment. CRI[4] was developed by Kubernetes and released in 2016 as an abstraction layer between a kubelet and a container runtime environment. A kubelet can communicate through the CRI with any container runtime environment without recompiling any component of a Kubernetes cluster. CRI implements the General-purpose Remote Procedure Call (gRPC) [5] framework to achieve this abstraction. The gRPC is an open-source RPC framework for high-performance applications, which was originally developed by Google. In addition to Kubernetes components, a container run time environment and a container network interface must be installed and configured correctly on every worker node. As mentioned earlier, we relied on the docker engine as a container runtime environment to deploy the artifact on a Kubernetes cluster. Finally, we relied on Flannel [6] to automate the implementation of local container network interfaces on the nodes. Flannel is a third-party open-source network configuration addon for Kubernetes [7]. It creates subnets on every node and facilitates communication between the running pods. Alongside the kubeproxy, it handles the communication between worker nodes and the master node.

Cluster operations management: Kubernetes offers a command-line tool named *kubectl*, which is used to control a Kubernetes cluster. It allows a user to communicate with a cluster control plane through the Kubernetes API. It is the main tool used to deploy applications, manage resources, inspect worker nodes, and monitor workload. To simplify

[4]https://kubernetes.io/docs/concepts/architecture/cri/ [Last checked 13.01.2024]
[5]https://grpc.io/ [Last checked 13.01.2024]
[6]https://github.com/flannel-io/flannel [Last checked 09.01.2024]
[7]https://kubernetes.io/docs/concepts/cluster-administration/addons/ [Last checked on 20.02.2024]

resources management of MESEAS for users, a Kubernetes dashboard[8] addon is deployed. Kubernetes dashboard is also a third-party open-source User Interface (UI) addon for Kubernetes [9]. The dashboard offers the user an overview of resources and full control of the deployment set of MESEAS components. It translates user decisions that are conveyed through the web interface to kubectl commands. These commands are then executed by kubectl and communicate with the cluster control plane.

Figure 4.4 depicts an example x86-64 physical node of MESEAS cluster using Kubernetes. The deployed dashboard allows the user to manage physical resources easily. It provides an overview of claimed resources in terms of CPU, main memory, and the number of pods running on every node, as presented in the figure. For instance, 21 containers are running on this node. It also provides information about network availability, memory pressure, storage pressure, etc. The state ready implies that the node is healthy.

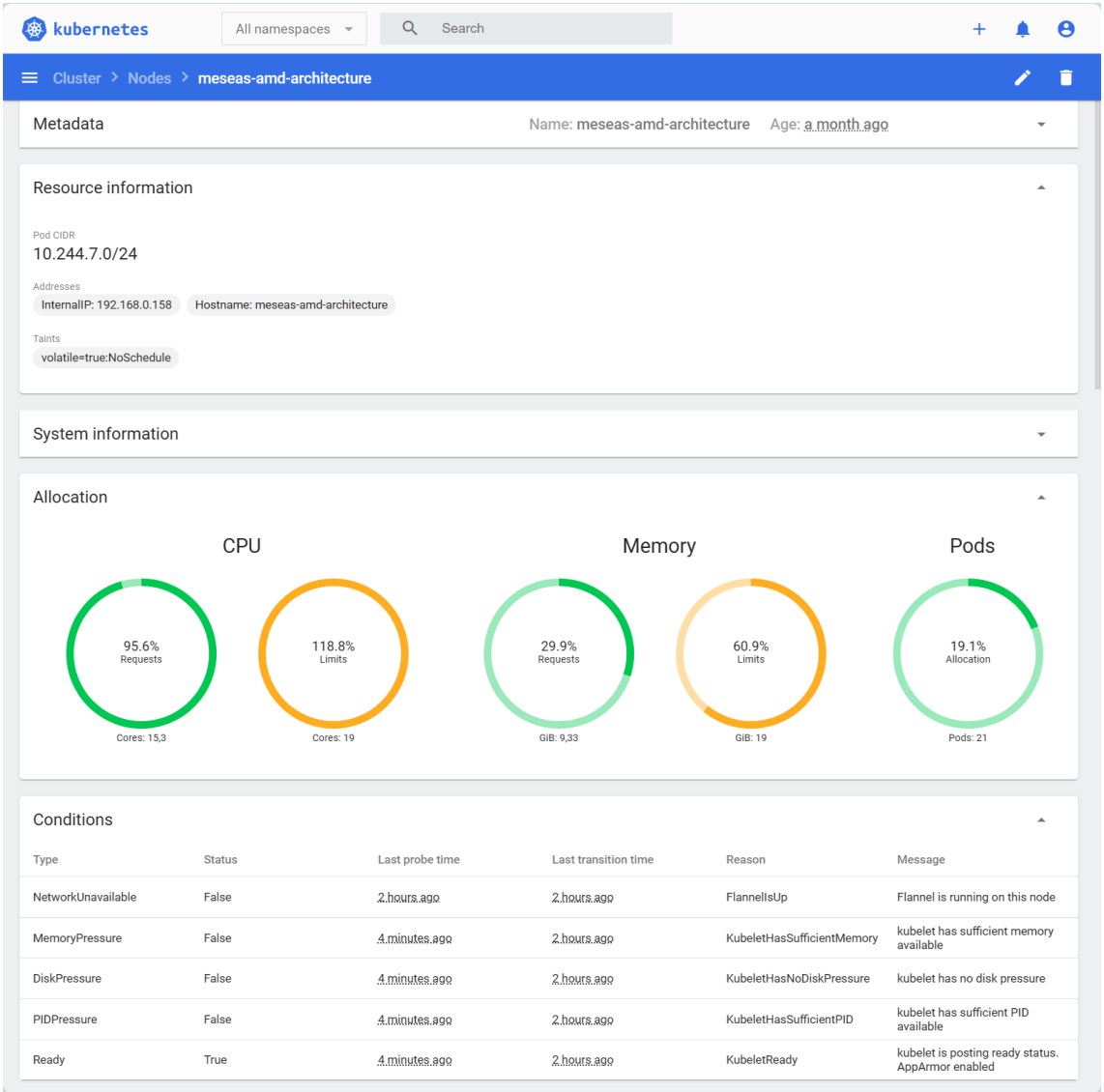


Figure 4.4: Example node of MESEAS composable deployment using Kubernetes.

[8] https://github.com/kubernetes/dashboard [Last checked on 20.02.2024]
[9] https://kubernetes.io/docs/concepts/cluster-administration/addons/ [Last checked on 20.02.2024]

We adopted Kubernetes for the deployment of the artifact since automation is essential. Kubernetes facilitates automated service discovery, scheduling, cluster orchestration, load balancing, storage management, and resource allocation. The physical resources required to host MESEAS components can be easily scaled up to reach high-quality solutions for a scheduling problem. After finding a solution, they can be scaled down. The health of physical resources can be monitored using the deployed dashboard to ensure quick recovery by restarting failed containers or replicating them automatically. This feature is essential if the scheduling method is utilized to address scheduling concerns in real-time. Kubernetes can be deployed on-premise, on the cloud, or on a hybrid infrastructure. The control plane of Kubernetes is intended to run on Linux, but within a cluster, applications can run on both Linux and Windows. At the time of writing, the latest version of Kubernetes is 1.29.

4.3.3 Component deployment and adopted frameworks

This section introduces the most significant open-source frameworks that are crucial for the execution and proper operation of the discussed subsystems of MESEAS methodology (cf. Section 3.3).

4.3.3.1 Simulation and heuristic subsystem

The first prototype of the artifact utilized the ExtendSim simulation package for modeling and simulation. However, the integration of external libraries was complicated. In addition, ExendSim can not be executed in a container and would require a major modification in the artifact's design. Therefore, an architecture decision is made to move to open-source simulation packages to develop the final prototype of the artifact. After the migration, the final design of the heuristic and simulation subsystem of MESEAS relies mainly on the Python runtime environment for the proper execution of the main components. It may utilize the Java runtime environment if the method is instantiated to use the CloudSim plus simulation package. We will shortly present the simulation packages that we employed during the research and the development of the presented artifact. ExtendSim, CloudSim Plus, and Salabim will be briefly discussed in chronological order in terms of their use.

Commercial simulation package: ExtendSim [10] is a propriety simulation software. It provides an extensive collection of toolkits for creating, executing, and troubleshooting various simulation models. The software also enables the development of custom models and interfaces for both continuous and discrete event simulation models. ExtendSim simulation package is often used to build simulation models for manufacturing and supply chain environments. The hierarchical modeling capabilities allow an engineer to model very complex environments, which may rely on custom simulation components. The package supports advanced graphical interfaces and 2D/3D animations to enhance the presentation

[10] https://extendsim.com/[Last checked on 01.02.2024]

quality. For developing custom logic or some algorithms, ExtendSim incorporates a built-in, compiled language called ModL (Krahl, 2007). Although it offers users flexibility in developing complex simulation models, it is tedious to integrate complex optimization or machine learning algorithms. ExtendSim can only be installed on Windows operating systems. At the time of writing, the latest stable version of ExtendSim is 10.1.0.

We relied mainly on the ExtendSim simulation package before migrating the artifact and implementing it using open-source technologies. The early conducted investigations on the potential of combining simulation, constructive heuristic, and improvement methods are all conducted using ExtendSim to instantiate the simulation component of MESEAS (Nahhas et al., 2017a; Nahhas et al., 2018a; Nahhas et al., 2018b; Nahhas et al., 2019a). The simulation package offers powerful tools for modeling complex cloud and manufacturing scheduling environments. However, integrating complex machine-learning methods was complicated. In addition, the package at the time of developing the early prototype did not support parallelization, which imposed a major drawback in computational performance. We are aware that multi-threading was introduced in the latest edition of ExtendSim 10.1.0, which we have not evaluated yet (cf. ExtendSim 10 [11]).

Java-based simulation package: We utilized the CloudSim Plus simulation package in the development of the simulation component. CloudSim is a Java-based discrete event simulator that was first proposed in 2009 by Buyya et al. (2009). CloudSim facilitates modeling, simulation, and evaluation of heuristic methods for scheduling problems in cloud environments. The official release of the CloudSim simulation toolkit was published a couple of years later with extensions, which allowed scholars to model and simulate virtualized environments in the cloud (Calheiros et al., 2011). The design of CloudSim is based on the GridSim package, which is an earlier work of the author presented to simulate Grid scheduling problems (Buyya and Murshed, 2002). Both simulation toolkits are based on the core simulation engine of the SimJava discrete event simulator that was presented by Howell and McNab (1998) in the late nineties.

To address some of the shortcomings of CloudSim, Silva Filho et al. (2017) presented the CloudSim Pluse simulation package. The CloudSim Plus is an extension of the CloudSim simulation toolkit, which presented additional functionalities to allow integration of various energy power models of computing servers in cloud environments (Silva Filho et al., 2017). CloudSim Plus [12] library includes various classes for defining cloud environments, applications, virtual machines, customers, computing assets, network infrastructure, and many others. The framework enables the advanced modeling of various cloud computing services, including infrastructure as a Service (IaaS) and software as a service (SaaS) models. It is widely utilized for evaluating and validating the performance of scheduling algorithms in cloud environments (Silva Filho et al., 2017). CloudSim Plus is an open-source package that is available on GitHub. To use the project, at least version 17 of the Java Development Kit (JDK) and Maven (3.8.6 or higher) must be installed. At

[11] https://extendsim.com/products/features/new [Last checked on 15.01.2024]
[12] https://cloudsimplus.org/[Last checked on 01.02.2024]

the time of writing, the latest stable version of CloudSim Plus is 8.5.0.

Throughout the development process of the prototype, CloudSim plus simulation toolkit has been utilized to model scheduling problems in cloud environments. For instance, during the integration of the simulation, heuristic library, and optimization components of MESEAS, we built simulation models using CloudSim Plus. We presented computational results demonstrating combining these techniques yields improvement in addressing scheduling concerns in cloud environments (Nahhas et al., 2021a). We further investigated various optimization criteria for solving scheduling problems in cloud environments using simulation models built in CloudSim Plus in (Remesh et al., 2022). We followed up on the previous investigation and utilized two large scheduling benchmarks of real workload in cloud environments using cloudSim Plus in (Remesh et al., 2023). The objective was to investigate and demonstrate achieved improvement in addressing scheduling concerns using the presented methodology (Remesh et al., 2023). The last two papers originated from a master thesis, which we closely supervised to study the performance of the integration for addressing large scheduling problems in cloud environments. The results are later published in Remesh et al. (2022) and Remesh et al. (2023).

Python-based simulation package: We relied on Salabim simulation packages written in Python to develop several modules in the simulation component. Salabim [13] is an open-source discrete event simulation engine. It was first developed for modeling and simulating complex logistics and manufacturing environments. The source code of Salabim is developed using Python programming language and thereby requires a Python runtime environment for execution. It can be included as a Python dependency, which allows for building and executing simulation models. The package provides queue handling, resource modeling, statistical sampling, and some monitoring features. It can be utilized in a broad spectrum of simulation applications, including warehouse simulation, manufacturing problems, or supply chain. It relies on some other Python dependencies and other visualization tools for building custom graphical interfaces for simulation models. For instance, it includes a built-in animation engine, which enables complex real-time 2D and 3D animation and advanced visual representations (Ham, 2018). Salabim is compatible with Linux, Microsoft Windows, and Mac OS X and runs on Python 2.6 or higher versions. At the time of writing, the latest stable version of Salabim is 23.3.13.

Deplyoment overview: During the research and development of the research artifact, we relied on Salabim simulation packages to develop the simulation component of the presented artifact. We also investigated and evaluated the integration of simulation component with different heuristic, optimization, and DRL methods (Nahhas et al., 2021b; Nahhas et al., 2022a; Nahhas et al., 2022b; Nahhas et al., 2024a). Figure 4.5 depicts the administration UI of MESEAS simulation and heuristic subsystem. MESEAS simulation and heuristic subsystem are deployed using a deployment set in Kubernetes. It allows users to scale up or down the number of running simulation instances by a few clicks.

[13] https://www.salabim.org/ [Last checked on 01.02.2024]

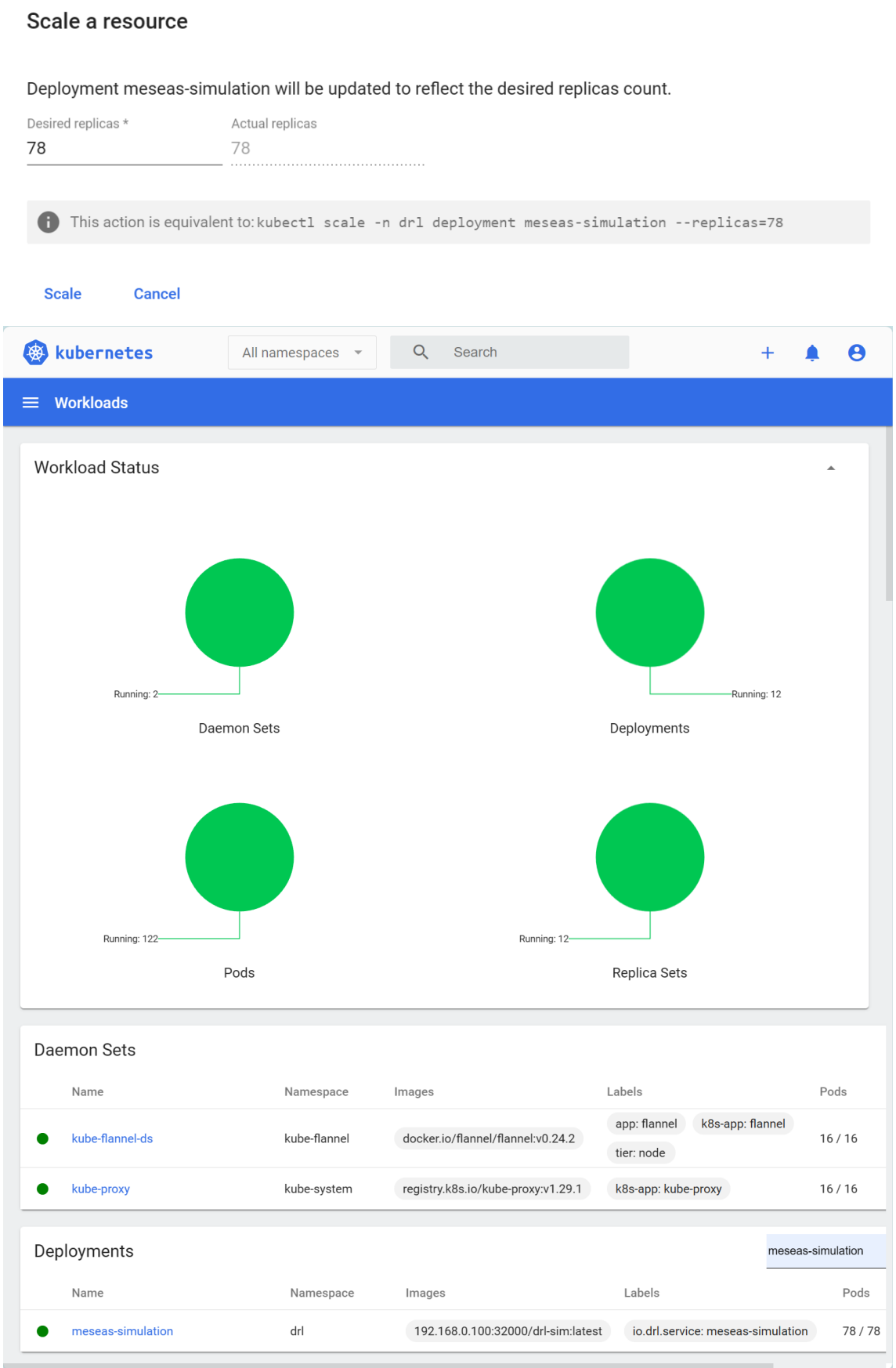


Figure 4.5: Deployment of MESEAS simulation and heuristic subsystem.

As shown in the upper part of Figure 4.5, the number of simulation workers can be scaled up or down depending on the problem at hand. In this example, 78 containers run simulation instances that evaluate suggested solutions by the optimization and machine learning subsystem. In the upper part of the figure, an overview of *"Daemon Sets"*, *"Deployments"*, *"Pods"*, and *"Replica Sets"* is demonstrated. *"Daemon Sets"* are used to deploy and manage the operation of services that must be present on all physical nodes in the cluster. For instance, the kube-proxy and the container network interface (Flannel) services must be running on all nodes. Therefore, we rely on daemon sets to deploy them. *"Deployment Sets"* are ideal for deploying containerized simulation and heuristic library components using a YAML file into *"Pods"*. YAML [14] abbreviation stood for (Yet Another Markup Language) and later changed to (YAML Ain't Markup Language). It is a data serialization language that was developed with a strong emphasis on human readability[15].

A YAML file in Kubernetes is a deployment manifest file that describes the type, the metadata, the container specification, the number of desired application replicas, and other deployment details. It is mainly used to roll out updates in the source code of the MESEAS method. Deployment sets in Kubernetes rely on Replica sets to manage deployment operations. *"Replica sets"* ensure that instances of an application are available in the system and automatically create new pods if some pods fail in the cluster. In this example, 122 pods are running in the cluster (cf. Figure 4.5). This number includes not only simulation instances but also optimization, database, messaging, and cluster management services. The green color implies that all services are healthy and running. Otherwise, a mix of green and red would be depicted, indicating the services that failed for further inspection.

4.3.3.2 Optimization and machine learning subsystem

The final prototype of MESEAS is developed using open-source technologies. Relying on open-source technologies facilitates easy integration of a state-of-the-art framework for developing optimization and machine learning components. In this section, we will highlight the most significant frameworks that we adopted to develop the final prototype of the presented methodology. For instance, the optimization component of MESEAS relies on the Distributed Evolutionary Algorithms in Python (DEAP) framework to adopt and implement evolutionary optimization algorithms. Meanwhile, we relied on the OpenAI Gymnasium framework to develop the DRL data structure required for training a DRL model. Ray Rlib is utilized to parallelize the training of multiple DRL models and merge their deep neural networks. To handle deep neural network training, we utilized TensorFlow. All mentioned frameworks are developed in Python. Therefore, the optimization and machine learning subsystem of MESEAS requires a Python runtime environment for the proper execution of its underlying components.

[14] https://yaml.org/spec/history/2001-08-01.html [Last checked on 20.12.2023]
[15] https://yaml.org/spec/1.2.2/ [Last checked on 20.12.2023]

Optimization framework: We utilized the Distributed Evolutionary Algorithms in Python to develop some modules in the optimization component. The DEAP[16] framework is a Python-based framework that offers interfaces and reference components to develop and adopt Evolutionary Algorithms (EA). As the name suggests, it focuses on evolutionary optimization methods and facilitates the investigation of various optimization approaches for solving combinatorial optimization problems. DEAP framework offers various libraries and practical tools for quickly developing customized evolutionary improvement methods (Fortin et al., 2012). DEAP's architecture comprises two basic components: a creator and a toolkit. The creator module allows the development and investigation of complex genotypes and maintains populations created based on the data structure of investigated genotypes. The toolkit offers data structure and required libraries to select and develop evolutionary algorithms and customize evolutionary operators such as selection, crossover, and mutation (Fortin et al., 2012). Finally, evolutionary methods require an evaluation method, which is developed using simulation techniques (cf. Subsection 3.4.4 and Subsection 3.6.2).

DEAP module is developed with a strong emphasis on parallelizing evolutionary improvement methods. It can be integrated with several parallelization technologies, such as Python-multiprocessing, which is adopted to develop the monolithic deployment of the presented methodology. In the coming subsections, we will briefly discuss parallelization technologies that have been adopted in various deployments. Since the module is developed in Python, it can be integrated with several machine-learning libraries like scikit-learn [17](Kim and Yoo, 2019, pp. 139–142). DEAP is compatible with Linux, Microsoft Windows, and Mac OS X and requires Python 2.6 or later versions. At the time of writing, the latest stable version of the DEAP is 1.0.2.

DRL application programming interface: To develop the DRL scheduling model, we utilized the Gymnasium data structure of (cf. Subsection 3.6.3 and Subsection 3.6.4). Gymnasium (formerly known as OpenAI Gym) is a Python-based standard API for reinforcement learning research and development. It supports a variety of reference DRL environments[18] that can be used to train DRL models. It offers many open-source implementations of reinforcement learning environments of different complexity. For instance, the reference implementations of several Atari games and 2D/3D robot control simulations have frequently been used to investigate new DRL approaches (Brockman et al., 2016). OpenAI data structures map the MDP notations to reference interfaces to develop observation and action spaces for custom RL and DRL methods. Relying on this framework certainly reduces effort, which must be invested in developing and evaluating new reinforcement learning approaches. It is a well-established framework in the research community for comparing the performance of new approaches against previous ones (Buduma and Locascio, 2017, p.254). In MESEAS, Gymnasium is integrated with other Python-

[16]https://deap.readthedocs.io [Last checked on 01.02.2024]
[17]https://scikit-learn.org/stable/[Last checked on 01.02.2024]
[18]https://www.gymlibrary.dev/index.html [Last checked on 23.12.2023]

based libraries like RLlib and TensorFlow to develop the machine learning component[19]. OpenAI Gym is compatible with Linux and MacOS and supports Python 3.7 and higher versions. At the time of writing, the latest stable version of Gymnasium is 0.29.0.

DRL training and distribution frameworks: To evaluate the developed DRL scheduling models, we utilized RLlib to adopt and implement several prominent DRL methods (cf. Subsection 3.6.3 and Subsection 3.6.4, respectively). RLlib [20] is an open-source Python-based RL framework for the development and deployment of RL and DRL models (Liang et al., 2018). The official documentation of RLib provides reference methods that are essential for implementing and executing RL and DRL components. RLlib depends on OpenAI Gymnasium APIs to define generic environment, action, state, and reward. The project is also well-established in academia and industry, which motivates scholars and practitioners to integrate libraries for several reference DRL algorithms such as DQN, PG, DDPG, PPO, and A3C. In the presented artifact, A3C and PPO are adopted to solve scheduling problems based on the presented DRL scheduling models.

Furthermore, RLlib facilitates the execution of distributed training and can be integrated with several multi-processing APIs, like Python multiprocessing[21] or RabbitMQ that are utilized for parallelization in MESEAS. As for efficient training of DRL models, it integrates with TensorFlow[22], which we relied on for training DRL models. TensorFlow offers a simple visualization tool called TensorBoard, which we utilized during the prototyping and development phases of the artifact. Relying on TensorBoard can be very beneficial to understanding the general behavior of machine learning models, which can be quite complex (Abadi et al., 2016). RLlib and TensorFlow are compatible with Linux, MacOS, and Windows (with beta version) and support Python 3.7 and higher versions. At the time of writing, the latest stable version of Ray is 2.9.0, and the latest stable version of TensorFlow is 2.15.0.

Deployment overview Figure 4.5 displays the administration UI of MESEAS optimization and machine learning subsystem. Similarly, deployment sets are utilized to deploy the optimization and machine learning components. These deployments are associated with corresponding replica sets to ensure that the required number of instances are up and running. For instance, in this example, we deployed 20 optimization instances to process requests of multi-optimization run experiments to solve a scheduling problem. The optimization instances are hosted on various physical nodes in the cluster and communicate with a set of simulation instances to evaluate generated candidate solutions. This number of available optimization instances can be scaled up or down based on the status of the experiments and available physical resources. As depicted in the upper part of the figure, the UI translates changes in the number of replicas to a command. This command is executed using the kubectl command line tool, which we discussed in Subsection 4.3.2.

[19]https://www.tensorflow.org/ [Last checked on 01.02.2024]
[20]https://docs.ray.io/en/latest/rllib/index.html [Last checked on 01.02.2024]
[21]https://docs.python.org/3/library/multiprocessing.html [Last checked on 08.01.2024]
[22]https://www.tensorflow.org/[Last checked on 08.01.2024]

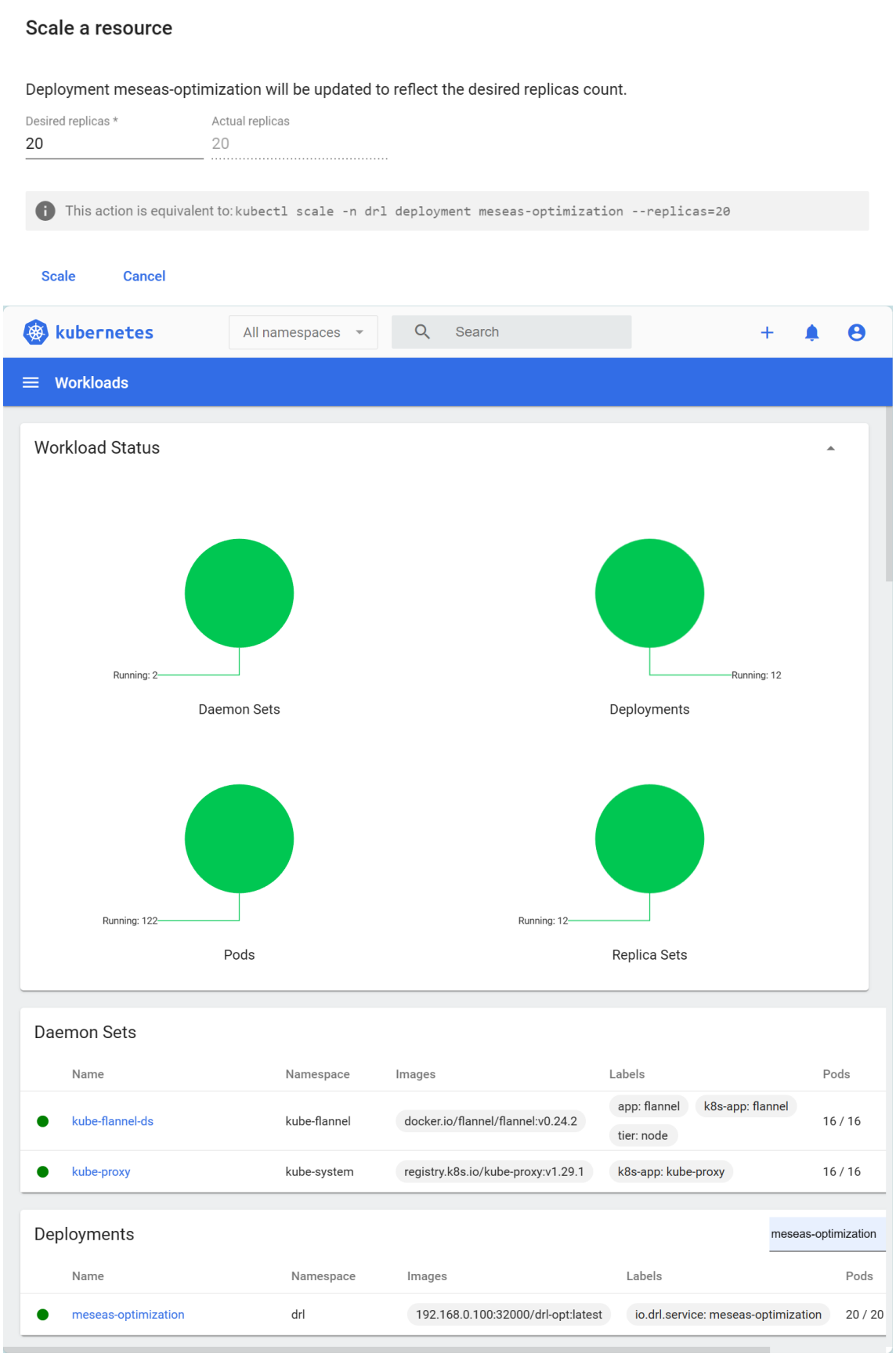


Figure 4.6: Deployment of MESEAS machine learning and optimization subsystem.

4.3.3.3 Data management and parallelization

To facilitate communication between the artifact's independent components, we relied, among other things, on MongoDB, RabbitMQ, and Kubernetes to implement the data management subsystem. MongoDB is utilized to deploy a database that is used to store data models and results. RabbitMQ is used to develop and implement several logging and messaging components to establish and maintain connections between the subsystems. Kubernetes scalability and workload distribution mechanisms are utilized to manage and distribute the workload on the infrastructure level. Since the core logical modules of MESEAS are developed in Python, we also developed the data management and parallelization components in Python. In this section, we highlight the most important ones.

Database: We relied on MongoDB to implement the database of the presented artifact. MongoDB [23] is a scalable open-source NoSQL database, which is well-suited for storing document data. It organizes data as collections of loosely structured tree-shaped documents using the JavaScript Object Notation (JSON) format (Botoeva et al., 2017), which we rely on for logging and results collection. The primary distinction between MongoDB and traditional Relational databases is the data storage structure. MongoDB uses Collections instead of Tables, JSON documents instead of Rows, and Embedding and Linking instead of Joins (Kanoje et al., 2015). Essentially, it operates like a file system. MongoDB is compatible with the main operating systems, such as Microsoft Windows, Mac OS X, and Linux. At the time of writing, the latest stable version of MongoDB is 7.0.

After several stress tests and storing a large number of detailed solutions during optimization, we concluded that MongoDB should be used only for data models, logging of key performance indicators, and best-found solutions. Detailed solutions may be stored and processed more efficiently using a traditional SQL database.

Messaging and parallelization: In the presented artifact, RabbitMQ is implemented as a round-and-robin load balancer between the subsystems. It acts as a parallelization layer and a broker between them. RabbitMQ[24] is an open-source message broker that facilitates asynchronous message-based communication across several applications, which is ideal for managing the communication between the simulation and optimization. It relies on the publish/subscribe (pub/sub) interaction architecture that enables the development of scalable and loosely coupled applications (Dobbelaere and Esmaili, 2017). RabbitMQ's primary components are publisher, consumer, exchange, and queue. A publisher (producer) is an application that generates a message with a certain topic and sends it via a message broker, eventually delivering it to consumers (subscribers). An exchange is a router that takes incoming messages from the publisher and routes them to queues based on certain rules. A message queue maintains messages and transfers them to message consumers (Sharvari and SowmyaNag, 2019).

[23] https://www.mongodb.com/[Last checked on 01.02.2024]
[24] https://rabbitmq.com/[Last checked on 01.02.2024]

To deliver messages, RabbitMQ relies on the Advanced Message Queuing Protocol (AMQP) - a standardized message brokering protocol. Pika[25] Python library is utilized to implement the AMQP protocol used by RabbitMQ to develop the logging and messaging system of MESEAS. In addition to AMQP, RabbitMQ supports several messaging protocols, including HTTP or WebSockets. Additionally, RabbitMQ offers a variety of important features like message persistence or delivery acknowledgment. These features are used to ensure the reliability of the combination of simulation and optimization subsystems. Missing results from simulation may comprise the validity of optimization or the training of a DRL agent. It also enables clustering, which allows several nodes to serve as a single message broker. That is essential for balancing workloads and scaling the system to manage a high volume of messages in which multiple instances of optimization and/or machine learning are running. It can be deployed on both cloud environments and on-premises and supports integration with other applications like Kubernetes, which makes it ideal for the deployment of the artifact. At the time of writing, the latest stable version of RabbitMQ is 3.13.0.

Figure 4.7 shows a parallelization comparison between Python multiprocessing[26], Apache Kafka[27], and RabbitMQ. The comparisons were conducted in a scientific student project supervised by the author. The main objectives were to investigate whether to keep the implementation of RabbitMQ or replace it with Apache Kafka and collect further insight for hyperparameters tuning of the optimization component. The baseline of the experiment was Python multiprocessing, depicted in red in the figure. We expected that a monolithic deployment of the artifact would at least slightly outperform a cluster or container-based deployment, assuming similar hardware characteristics. The optimization experiments were conducted with 100 population size and 500 generations to solve a scheduling problem. The same optimization experiment was executed using different numbers of available physical cores, as shown in the horizontal axis of the figure.

As expected, parallelization significantly accelerates the optimization process. In this example, the evaluation of 50000 solution individuals dropped from roughly 70 minutes to less than seven minutes. The experiments also showed that RabbitMQ implementation clearly outperformed Apache Kafka in our setup and sustained a marginal edge against Python multiprocessing, contradicting our expectations. The collected results were also used to decide on population size and number of generations hyperparameters of the optimization component. Increasing the number of simulation instances to more than 24 yielded rather a marginal improvement. However, further experiments showed that the utilization rate of the simulation instance can be significantly increased if we use a larger population size and reduce the number of generations. Therefore, the conducted experiments on the use case presented in the coming section are parameterized with 250 population size and 200 generations. The analysis included investigating several other messaging systems, and the obtained results are being prepared for publication.

[25]https://pika.readthedocs.io/en/stable/ [Last checked on 01.02.2024]
[26]https://docs.python.org/3/library/multiprocessing.html [Last checked on 11.01.2024]
[27]https://kafka.apache.org/ [Last checked on 11.01.2024]

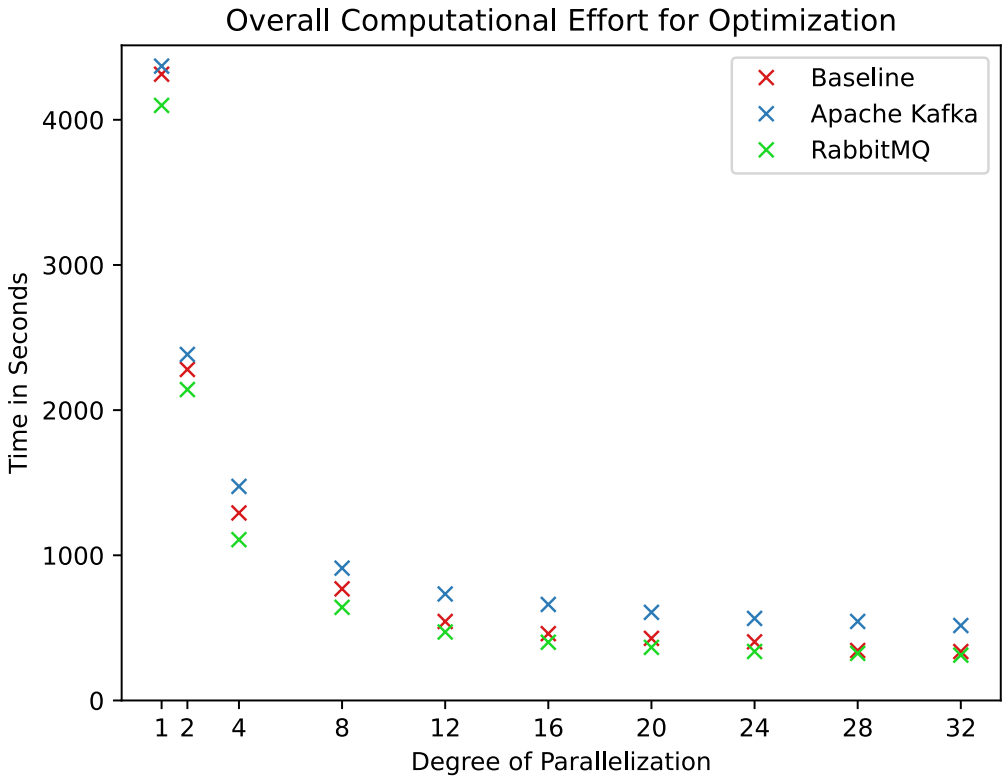


Figure 4.7: Comparison between various parallelization techniques.

Workload management and scalability: The presented artifact is developed with an emphasis on modular design and scalability. In the previous chapter, we extensively discussed the aspects of modularity in the design of MESEAS. Modularity is essential to meet functional and non-functional requirements, including scalability (cf. Section 3.1, Section 3.3). Scalability and parallelization of the presented methodology were implemented using open-source technologies and frameworks, such as RabbitMQ, Python multi-processing, Ray RLlib, or Kubernetes. To support a multi-architecture execution of the artifact, we relied on virtualization and containerization techniques for abstracting the running modules of MESEAS from the underlying hardware.

Kubernetes was a very suitable technology for deploying and operating the artifact in a cluster. It facilitates the utilization and management of heterogeneous infrastructures and offers easy integration of cloud resources to achieve scalability. Kubernetes technology stack includes several fundamental features to automate the cluster's scalability and configuration management. It is utilized to distribute and manage the MESEAS workload on an infrastructure level using deployment sets, replica sets, daemon sets, node labels, taints, and priority groups. Node labels can be used to restrict scheduling containers that do not match the stated label. Labels are expressed under container specifications using nodeSelector. The cluster scheduler allocates pods of some deployment only to those nodes that match the defined label/s (e.g., arm64-sim for allocating only simulation in-

stances to ARM64-nodes). Taints are used to refrain containers from being scheduled on some nodes. Tainting a node with *"control plane"* would require explicitly defining a corresponding toleration for it, so a pod is allocated to the node with this taint.

For instance, Figure 4.8 depicts an example in which a user is notified that the available physical resources in the cluster cannot host all requested containers of the simulation, optimization, and machine learning components.

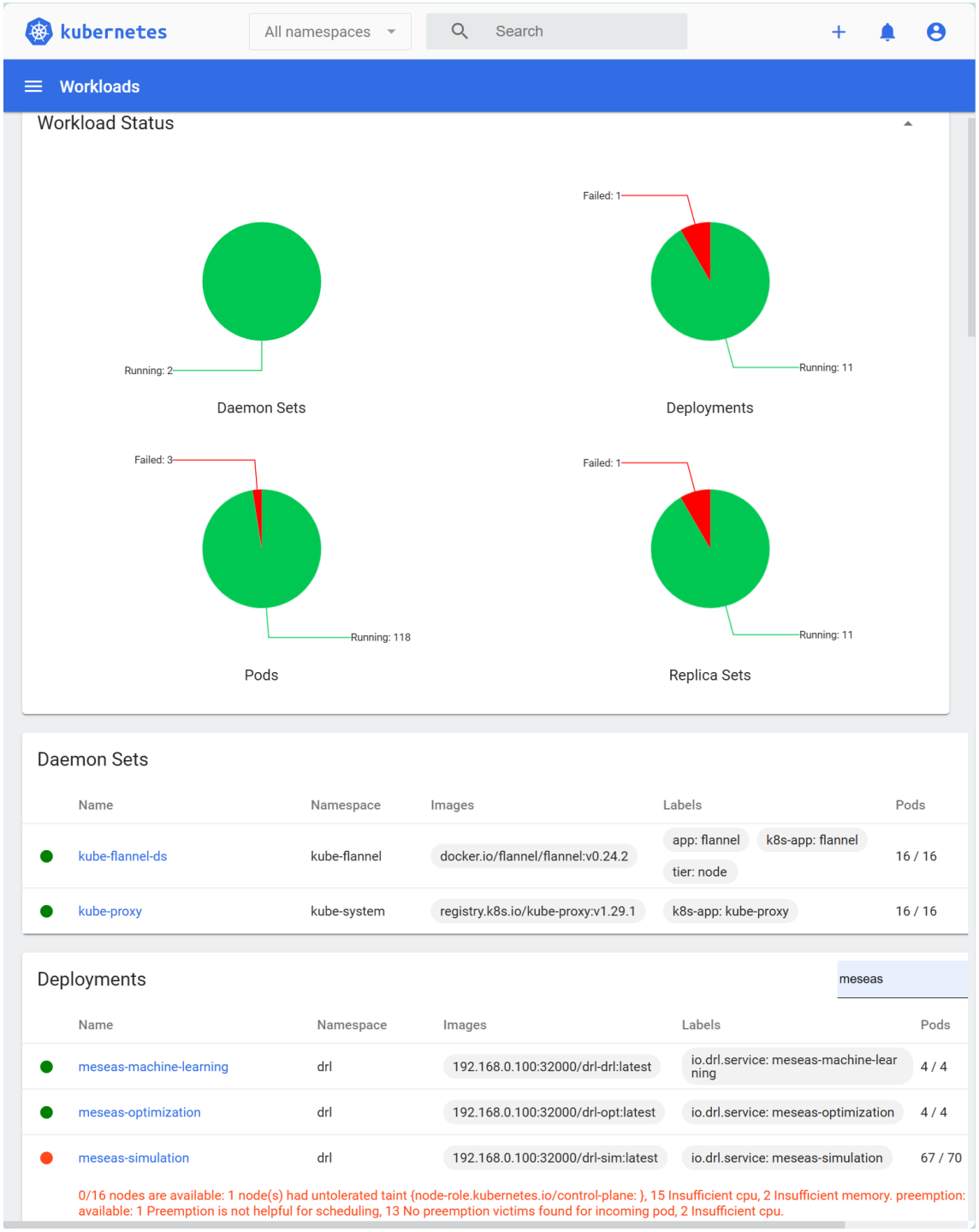


Figure 4.8: Parallelization and workload management.

In this figure, 118 containers are running in the cluster, and three pods failed to be scheduled, as shown in red. The administration UI also specifies which corresponding deployment and replica sets are not fully operational. In MESEAS, priority groups are defined to prioritize containers of different subsystems. For instance, network interfaces and proxies have, by default, the highest priority. Data management services have the second highest priority. The optimization and machine learning services are assigned to the third priority group. Finally, simulation subsystem services belong to the fourth priority group.

In the example presented in Figure 4.8, three simulation pods failed to be scheduled. In turn, *"meseas-simulation"* was depicted in red with its corresponding deployment and replica sets. The message shows that 15 physical nodes did not satisfy the requested hardware requirements of the service. It also demonstrates that there was a node that could satisfy the hardware requirements by taint with *"control plan"*. Without the toleration, the containers could not be scheduled to this node. In MESEAS deployment, the master node is tainted with *"control plan"* to ensure that nothing impacts its operational stability. It is used to host only cluster management-related services. Priority groups also enforce the scheduling of higher-priority services in case of resource scarcity. For instance, if we scale the number of required services by the optimization and machine learning subsystem from four to five each, the scheduler will evict lower-priority services to secure hardware resources. In this case, it will stop simulation containers and evict them.

4.3.4 Summary of the proof of concept (Eval 3.1)

In DSR, the implementation and instantiation of the artifact design are crucial to developing suitable and applicable IT artifacts that are relevant for organization (von Hevner et al., 2004, p. 77). Sonnenberg and vom Brocke (2012, p. 393) detailed the third evaluation activity and highlighted *"proof of applicability"*, suggesting various methods for evaluation. We relied on *demonstration with a prototype* and *experiment with the prototype* evaluation methods (cf. Subsection 3.5.3, Subsection 3.4.5, and Subsection 3.6.5). This section provided a brief overview of the instantiation of the presented artifact as a proof of concept. The presented prototype demonstrates that the presented design blueprints of MESEAS components can be integrated to fulfill the requirement specifications (cf. Section 3.1 and Section 3.3).

The developed artifact can be deployed and executed on multi-architecture and heterogeneous infrastructure as demonstrated by the prototypical instantiation (cf. Subsection 4.3.1). To achieve this objective, proper isolation and abstraction technologies are adopted to virtualize and containerize the components of the artifact. The parallelization and scalability of the presented prototype are achieved using cloud-native technologies such as Kubernetes and RabbitMQ, both of which require a resilient and modular design of the artifact to be adequately adopted. Automation of configuration management and cluster operation is accomplished by adopting various features of these technologies, as demonstrated in the overview of the deployment of each subsystem of MESEAS. Fi-

nally, we adopted open-source technologies and frameworks to develop and integrate the discussed simulation, heuristic, improvement, and machine-learning methods. Thereby, the researched and adopted set of technologies is *suitable* suitable for the development of the prototype. The prototypical implementation demonstrated that the adopted parallelization and cloud-native technologies yielded an *effective* and *efficient* operation of the artifact.

4.4 Evaluation in cloud environments (Eval 3.2)

4.4.1 Use case overview

In cloud environments, we presented an overview of workload scheduling strategies in the literature. We investigated the performance of heuristics methods in a use-case relying on real-world workload patterns of 290 SAP applications (Nahhas et al., 2018a). The workload patterns of the virtualized SAP application systems are used to model the system's behavior. Jobs or Virtual Machines (VMs) were scheduled to machines using heuristics to minimize energy consumption while considering performance concerns. We relied on discrete event simulation methods to build a simulation model. Then, we extended the analysis to investigate and evaluate the integration heuristic, simulation, and improvement methods for addressing scheduling concerns in the investigated environment (Nahhas et al., 2019a).

In cloud environments, switching idle machines to hibernation mode can significantly reduce the system's energy consumption (Beloglazov et al., 2012b, p. 759). An idle machine can consume, on average, 70 % as much energy as a highly loaded machine (Beloglazov et al., 2012b, p. 759). Scheduling and rescheduling heuristics must be applied to consolidate workload automatically. The included heuristics are combined with rescheduling mechanisms based on thresholds (Beloglazov et al., 2012a, p. 1410). Scheduling in real-time usually leads to partly satisfactory results. Hence, rescheduling mechanisms using decision points or rescheduling triggers to adapt to changes in the considered environment are suggested by scholars to achieve better results (Sun et al., 2024, p. 174). Rescheduling implies that a subset of jobs may be rescheduled on different machines to minimize some objective values. In cloud environments, rescheduling and live migration are interchangeably used terms to describe the reallocation of resources. However, rescheduling jobs leads to a degradation in performance, which must be minimized (Beloglazov et al., 2012b, p. 760). Finding a reasonable trade-off between these two conflicting objective values was the core subject matter of this use case. Based on the characteristics of the environment under investigation, we will instantiate the methodology for addressing scheduling concerns.

4.4.2 Methodology instantiation and problem formulation

Based on the formalization of the presented methodology in Subsection 3.4.3, MESEAS was instantiated to address scheduling concerns of the discussed use case as presented in

Equation 4.1. To address scheduling concerns in this environment, we utilized the first three functionality layers of MESEAS. The scheduling data model, the simulation, the heuristic library, and the optimization components are therefore instantiated. Given the scheduling problem SP_{single}, we utilize $MESEAS_{(X,\ T)}$ to find a solution $X \in \mathbb{X}$ that minimizes the objective function Γ_ω during the scheduling period T. In the course of this section, we will elaborate on the instantiation of the methodology.

$$MESEAS_{(X,\ T)} = \langle SP_{single} \mid PI_{single} \mid \Gamma_\omega \mid Sim \mid HL \mid Opt \rangle \tag{4.1}$$

The scheduling problem - $\mathbf{SP_{single}}$

The investigated cloud environment contained heterogeneous physical servers with various resource capacities. We investigated a subset of the physical infrastructure. All physical machines can process all types of jobs with no constraints. Jobs are released in the system for processing and then leave the system after completion based on defined mathematical distributions. Therefore, the investigated system was subject to release time constraints. The system can be accordingly expressed as a single-stage scheduling system $\mathrm{SP}_{\mathrm{single}}$. Jobs can be processed or hosted by a physical machine only if the machine can fulfill the computing requirements. Hence, the scheduling environment considered is subject to machine capacity constraints. In summary, scheduling jobs to resources with respect to several objective values and constraints listed in the following bullet points:

- Release date $\beta_1 = r_j$: Jobs are released for scheduling based on their associated release time during the scheduling period T.
- Priority families $\beta_2 = D_{pr}$: Jobs belong to various priority families that are based on the defined Service Level Agreements (SLA).
- Machine capacity constraint $\beta_3 = M^C$: Jobs are scheduled on physical machines only if a machine fulfills its requested computing requirements (cf. Section 2.1.2, Equation 2.1).
- The total energy consumption $\gamma_1 = E$: The energy consumption of computing resources is a significant cost and sustainability driver in a cloud environment. Therefore, scheduling problems in cloud environments are solved by minimizing overall energy consumption. It can be minimized by reducing the total number of online hours and maximizing the number of hibernation hours of machines.
- The total number of penalties $\gamma_2 = U$: SLA violations cost loss of reputation and are associated with financial penalties. Rescheduling jobs to reduce energy consumption may lead to violations in SLAs. Therefore, solution methods for scheduling problems in cloud environments must be designed to take into consideration the minimization of total penalties by minimizing the number of migrated jobs.

Given the discussed description of the considered cloud environment, the scheduling problem is expressed by $\mathrm{SP}_{\mathrm{single}} = \langle Q_m |\ r_j,\ M^C,\ D_{pr} \mid E,\ U \rangle$.

Problem instance - $\mathbf{PI_{single}}$

The considered cloud environment was characterized by known workload patterns that can be described using mathematical distributions. We utilized triangular mathematical distribution to model the job release time for scheduling in the system and their processing time. In this environment, 290 jobs belonging to several priority families may be released for scheduling in the system. After a job is processed based on the workload pattern, it leaves the system. During a scheduling period, many jobs sharing the discussed characteristic may enter the system. Based on the data structure presented in Section 3.4.2, the instantiation is summarized in the following bullet points:

- The set $T = \{T_t, \ldots, T_{|T|}\} : \forall\, t \in \{1, \ldots, 24\}$ denote a day scheduling period divided per hour. Given a discrete change in time horizon $\Delta T_1 = 1$, we move from T_1 to T_2.
- The set $J = \{J_j, \ldots, J_n\} : \forall\, j \in \{1, \ldots, 290\}$ denote the number of jobs which are released for scheduling during a day of operation T based on the workload pattern.
- The set $M^C = \{M_i^C, \ldots, M_m^C\} : \forall\, i \in \{1, \ldots, 8\}$ denote the machines that are available to process jobs in parallel.
- The $p_{i,j} \in \mathbb{R}^+$ denote the required processing time of a job $J_j \in J$ to be hosted and processed by a machine $M_i^C \in M^C$ and is defined based on workload pattern.

Objective function - $\mathbf{\Gamma_\omega}$

Based on the presented instantiation of the scheduling problem and the description of the problem instance, the scheduling problems were solved subject to the minimization of the objective function in Equation 4.2. The problem instances are solved by finding a solution $X \in \mathbb{X}$ that minimizes the objective values. The persued solution X repsents a map that schedule the jobs $J = \{J_j, \ldots, J_n\} : \forall\, j \in \{1, \ldots, 270\}$ to the machines $M^C = \{M_i^C, \ldots, M_m^C\} : \forall\, i \in \{1, \ldots, 8\}$. The overall objective function is formulated using a weighted sum approach such that the set $\Gamma = \{\gamma_1,\ \gamma_2\}$: subject to the set of weights $\omega = \{\omega_1 = 0.3,\ \omega_2 = 0.7\}$ as presented in Equation 4.2.

$$\arg\min_{X \in \mathbb{X}} \Gamma_\omega(X) = \arg\min_{X \in \mathbb{X}} \left(\omega_1 \cdot \left(\sum_{i=1}^{m} \gamma_1(X) \right) + \omega_2 \cdot \gamma_2(X) \right) \tag{4.2}$$

Simulation - (Sim)

For solving the scheduling problem, simulation methods are combined with heuristic and optimization methods. We utilized the ExtendSim simulation package to build the simulation model.

Heuristic Library - (HL)

We developed two simple constructive heuristics, namely, the Energy-Aware Family-Fit-Decreasing (EA-FFD) and the Workload-Aware Family-Fit-Decreasing (WA-FFD) . The

rationale of the EA-FFD was presented and thoroughly discussed in Algorithm 11. The logical design of the WA-FFD is significantly similar except for taking the workload of machines into account instead of their energy consumption. Available machines are sorted in a decreasing order to consolidate the workload of active machines. Instead of scheduling a job based on the minimum energy consumption, it is scheduled on a machine with the maximum workload. Both algorithms are included with three rescheduling upper and lower thresholds, resulting in six different allocation heuristics. The upper and lower bounds of the algorithms trigger rescheduling processes in the considered system to adjust scheduling. The following points summarize the utilized heuristics:

- A_1: EA-FFD with 90 % upper threshold and 10 % lower threshold.
- A_2: EA-FFD with 80 % upper threshold and 20 % lower threshold.
- A_3: EA-FFD with 70 % upper threshold and 30 % lower threshold.
- A_4: WA-FFD with 90 % upper threshold and 10 % lower threshold.
- A_5: WA-FFD with 80 % upper threshold and 20 % lower threshold.
- A_6: WA-FFD with 70 % upper threshold and 30 % lower threshold.

Optimization - (Opt)

The optimization component of MESEAS was instantiated using the indirect discrete encoding model. The discussed allocation heuristics markers were utilized to define the genotypes for the implemented Genetic Algorithms. Based on the instantiation of the HL, the set $A = \{A_a, \ldots, A_{|A|}\} : \forall\, a \in \{1, \ldots, 6\}$ denoted a set of six allocation heuristics. The genotypes were grouped into a chromosome represented by a vector Φ^A as presented in Equation 4.3. Each genotype Φ_φ initialized an allocation constructive algorithm A_a, which was used at the decision point T_φ. According to this encoding, the allocation algorithm may be switched every hour based on the results of the optimization.

$$\Phi^A = \Big[\Phi_\varphi, \quad \cdots, \quad \Phi_{|T|}\Big] : \quad (\varphi = 1, 2, \ldots, 24) \wedge (\Phi_\varphi = 1, 2, \ldots, 6) \tag{4.3}$$

4.4.3 Computational results

Component configuration

The experiments relied on the scheduling data model, the simulation component, the heuristic library component, and the optimization component. The investigation instantiated the single-population configuration of the optimization component. A weighted sum approach was adopted to weight the objective values. Conventional GA was used in the optimization. We utilized a two-point crossover operator with $\sigma = 0.6$ crossover probability to conduct the evaluation. The two-point crossover evolutionary operator is widely adopted in the design of evolutionary methods. It is supported by a fundamental theoretical foundation presented by (Holland, 1975) and further empirically investigated to

yield good performance by de Jong (1975, pp. 20-21). We used a shuffling index mutation function with a probability $\delta = 0.5$ to maintain diversity in the population. This mutation function was first introduced by Eshelman et al. (1989, p. 15) to mitigate crossover performance and diversity issues in evolutionary methods. We parameterized the GA with a population size $\mu = 100$. We used the number of generations as a breaking condition with a value $maxGen = 100$.

Analysis of computational results

The overall scheduling period considered was 120 hours, which corresponds to five days. To evaluate each solution individual of GA, we collected results from 10 simulation runs before assigning normalized fitness values. Jobs in the considered environment are released for scheduling using stochastic interarrival distribution. Hence, after the optimization was completed, we conducted a simulation experiment to collect results from 200 simulation runs using the final solution. The objective was to establish confidence in the quality of the solutions using a 95 % confidence interval and measuring the margin of error for possible deviations. In this use case, the set of heuristic indexes suggested by the optimization component and the schedule after decoding are important parts of the solutions.

Figure 4.9 depicts the final results of the investigated heuristic and the optimization component using the allocation encoding model (MESEAS-Single). The left part of Figure 4.9 (A) displays the results of the individual heuristics and the optimization in minimizing the total number of online hours of all machines. The central part of the Figure 4.9 (B) demonstrates the total number of initiated rescheduling processes. The total number of initiated rescheduling has an impact on the second objective value and provides further insights into the operational stability of the considered environment. The right part of Figure 4.9 (C) shows the results of the heuristics and the optimization in minimizing the total number of rescheduled jobs.

The computational results showed that heuristic methods could, on average, achieve up to 73 % energy saving compared to is-situation in the considered environment since load consolidation strategies were not utilized. With 253 total online hours, the WA-FFD reported a 73.6 % reduction in the overall online hours. The overall online hours of all machines during the considered scheduling period is 960 hours. Every machine has 120 online hours during five days of operation. It slightly outperformed the other EA-FFD and a combination of the algorithms using MESEAS-Single. Second, MESEAS-Single minimized the total online hours to 272, amounting to a 71.7 % reduction. The EA-FFD achieved solutions where 295 online hours of machines were required to process jobs using 69.3 % of the overall capacity. Looking into the central part of the figure, one could assume, before looking into the total number of rescheduled jobs, that the EA-FFD with 22 initiated rescheduling processes impacts most the stability of the system. However, the results in terms of total rescheduled jobs showed that the WA-FFD performed the worst, with over 342 rescheduled jobs. The EA-FFD reported 275 rescheduled jobs during the considered scheduling period with 19.6 % outperformance against the WA-FFD.

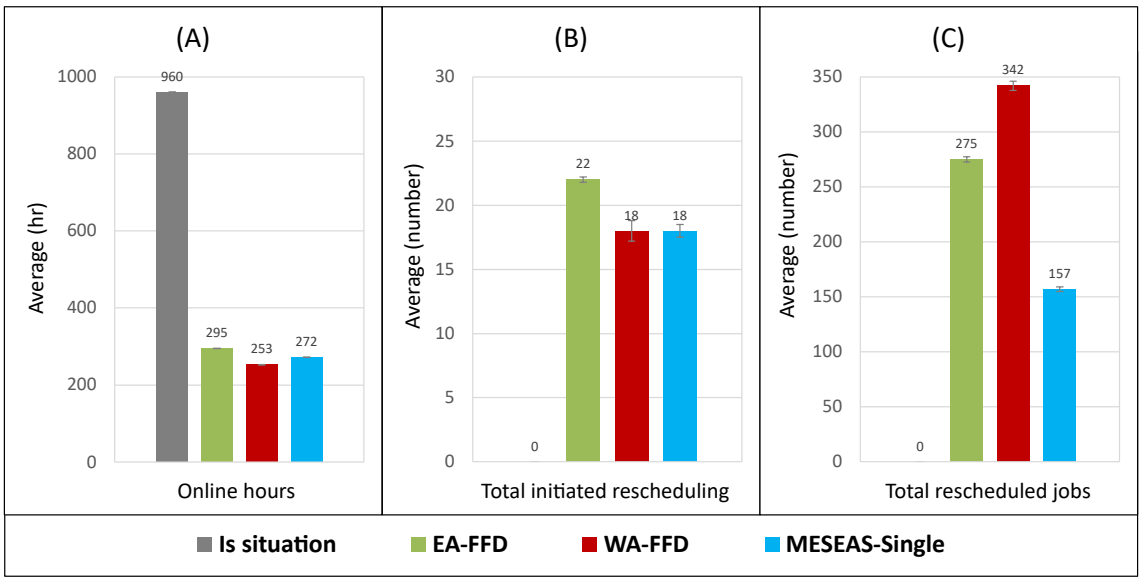


Figure 4.9: Average of online hours and rescheduled jobs during the scheduling period modified based on Nahhas et al., (2019a).

Finally, MESEAS-Single significantly outperformed both heuristics in minimizing the total number of rescheduled jobs. It achieved 54.1 % better results than the WA-FFD and 42.9 % than EA-FFD in minimizing the number of rescheduled jobs. With a 95 % confidence interval, the recorded margin of error in minimizing the total number of rescheduled jobs of the investigated algorithms ranged between ±2.07 and ±5.56. Similarly, the analysis of the results using the same confidence interval showed an error margin ranging between ±1.82 and ± 4.29 in terms of the total online hours.

The previous results demonstrated that combining various allocation heuristics during a scheduling period yields significant improvement in terms of minimizing the total number of rescheduled jobs. The overall energy consumption is investigated by minimizing the total number of online hours, which maximizes the total number of hibernation hours. The recorded results emphasized the potential of scheduling techniques to reduce overall energy consumption with an improvement that amounted to 73.6 %. In comparison to WA-FFD, the results of MESEAS-single indicated that sacrificing 1.9 % minimization of online hours can minimize up to 54.1 % of rescheduled jobs. In summary, the collected results demonstrated that combining several heuristics over the scheduling period yields better overall performance than their utilization individually. Hence, the proposed approach demonstrated that it adapted to the workload of the system during the scheduling period. It suggested switching between several heuristics to suit the workload pattern at every decision point, which indicated better objective values.

To avoid repetition, we will summarize the preliminary evaluation of the presented methodology to address single-stage scheduling problems in two further use cases. The instantiation and utilization of the presented methodology were almost identical except for a slight deviation in the number of used heuristics. The objective was to establish comparability with related works using the same workload traces (Moges and Abebe, 2019;

Beloglazov et al., 2012a; Beloglazov et al., 2012b). Further details of the instantiation and collected results can be found in Appendix B.

4.4.4 Instantiation for further use cases

In the second use case in cloud environments, combining simulation, heuristic, and optimization methods was investigated for solving larger problems using the well-known PlanetLab problem instances (Park and Pai, 2006, pp. 68-69). The academic community widely utilizes these problem instances to evaluate the performance of scheduling methods. For instance, several scheduling methods are evaluated using the setup of PlanetLab workload traces (Moges and Abebe, 2019, p. 6; Witanto et al., 2018, p. 39; Beloglazov et al., 2012a, pp. 1414-1415). The PlanetLab problem instances were collected in the context of the ConMon project, which was dedicated to monitoring the workload of PlanetLab infrastructure (Park and Pai, 2006). The PlanetLab infrastructure contained servers located in over 500 locations (Beloglazov et al., 2012a, p. 1415). The workload of thousands of virtual machines running on these servers was monitored, and workload traces were collected. The system is essentially a single-stage scheduling environment. Similar to the discussed use case, the problem was solved using a weighted sum approach to minimize the overall energy consumption, the SLA violations, and the total number of migrated virtual machines. A weighted sum formulation of the objective function transformed the problem into a mono-objective optimization problem. The initial results were published in a master thesis (Cheyyanda, 2020), which the author of this thesis supervised. For evaluation, ten problem instances from workload traces of the PlanetLab benchmark were utilized.

CloudSim Plus was utilized to build a simulation model of the defined cloud environment. The conducted experiments were compared to the scheduling heuristic presented in related works (Moges and Abebe, 2019; Beloglazov et al., 2012a; Beloglazov et al., 2012b). Eight allocation heuristics presented in these works were utilized and integrated with the optimization. The included allocation heuristics were used as a marker to define the genotypes for the implemented GA. Therefore, the single-population encoding of MESEAS was sufficient to investigate the problems. The experiments were subject to the same conditions and problem formulation presented in (Beloglazov et al., 2012a) and (Moges and Abebe, 2019) to ensure comparability. The performance of the scheduling methodology was compared to well-established heuristic techniques in the literature (Beloglazov et al., 2012a; Moges and Abebe, 2019). The collected results demonstrated that the presented methodology achieves the best results in minimizing energy consumption with a minimum 30 % reduction and up to 47 % compared to other techniques. It also sustained an acceptable level of SLA violations with a slight increase of 0.04 % since both objective values are conflicting in nature (Nahhas et al., 2021a).

In the second use case, further evaluation of the proposed methodology for solving single-stage job scheduling problems in cloud environments was conducted using the NASA and KTH workload traces (Dror G. Feitelson et al., 2014, p. 2970). The NASA and KTH

workload traces are often adopted to evaluate the performance of scheduling techniques in cloud environments (Bandaranayake et al., 2020). The initial results were published in a master thesis (Remesh, 2021), which the author of this work supervised. The study's second objective was to investigate the technique's performance given various combinations of objective values such as the makespan, the average waiting time, the throughput, and the average flow time.

CloudSim Plus framework was also used to build a simulation model to investigate the workload traces of the cloud environments. The computational results suggested that combining heuristics with improvements methods performed best when the objective function was to minimize the average flow time and the average waiting time (Remesh et al., 2022). To conclude the evaluation, a focused evaluation of the presented scheduling method using the identified combination of objective values was presented in (Remesh et al., 2023). Based on the same experimental setup, the obtained computational results suggest a complete dominance of the presented method compared to the baseline heuristic techniques for solving the problems in terms of minimizing the makespan, the average flow time, the average waiting time, and the maximization of the throughput. A detailed analysis of the computational results shows a consistent increase in improvements achieved with the presented scheduling methodology compared to other heuristics, with an increase in the number of jobs that must be scheduled. Further details can be found in Appendix B.

4.4.5 Summary (Eval 3.2)

In conclusion, the conducted experiments in the first cloud environment showed that MESEA-single leveraged the suitability of different heuristics under different workload conditions in the considered environment to achieve better performance. The experimental results contribute to the evaluation of the discussed hypothesis and answer a research question (cf. Hypothesis 2 and hypothesis 3). The investigation was conducted on a part of the considered scheduling environment to focus on measuring the potential of combining different heuristics to deal with scheduling concerns. Evidently, larger scheduling environments are more complex, resulting in more complicated optimization problems with higher optimization potential. The computational effort becomes a tedious challenge when the complexity of the considered problem increases. Our experiments in this use case were conducted with no parallelization since we relied on a propriety simulation package at that time. For proper deployment of the presented methodology and full integration with real systems, migrating to open-source solutions was necessary to utilize proper parallelization techniques.

The results of the conducted evaluation in the second use case using the PlantLab benchmark are consistent with those of the first use case. Utilizing several allocation heuristics over a scheduling period yielded better overall results in terms of minimizing energy consumption while considering performance measures (Nahhas et al., 2021a). The experimental results were also dedicated to establishing a comparison to related works presented by Beloglazov et al. (2012a) and Moges and Abebe (2019). Finally, the findings

of the third use case were also consistent with those of its predecessors (Remesh et al., 2022; Remesh et al., 2023). The conducted experiments in the second and third use cases lack statistical analysis and are experimental. Hence, the presented evaluation using the PlanetLab, NASA, and KTH workload traces meets, at best, the requirements of the third evaluation activity as suggested by (Sonnenberg and vom Brocke, 2012).

Scheduling environments are usually subject to changing workload conditions, which makes it difficult to find or develop a single algorithm that always suits workload patterns. The previously discussed results demonstrated significant potential improvement that can be achieved through combining simulation, heuristic, and improvement methods for addressing single-stage scheduling problems. In general, single-stage scheduling problems are essentially not as complex as multi-stage scheduling problems. Therefore, we will focus on presenting and critically discussing the multi-stage evaluation of MESEAS components in the next evaluation.

4.5 Evalution in manufacturing (Eval 3.3 - partial Eval 4)

4.5.1 Use case overview

The presented artifact was evaluated for solving multi-stage scheduling problems in a Printed Circuit Board (PCB) manufacturing environment. PCB manufacturing environments are usually designed with similar features to assembly production environments (Smed et al., 2003, pp. 3-4). A PCB is manufactured after several processes are completed. In the investigated manufacturing environment, the manufacturing of a PCB is subject to completing at least the first two operations and may require up to four operations. In essence, it is a four-stage hybrid flow shop manufacturing system. In the first processing stage, PCBs are mounted with parts using Surface Mounting Devices (SMD) machines. The SMD processing stage is the most significant stage for the overall efficiency of the production environment due to the characteristics of such systems (Csaszar et al., 2000, p. 125). Machines in the SMD stage are configured according to the part types of the processed PCBs. After completing the mounting operations, PCBs undergo a quality control process using Automated-Optical-Inspection (AOI) machines. Some PCBs may undergo the Selective Soldering (SS) process, where some parts are mounted independently on the boards. Finally, some PCBs are processed by Conformal Coating (CC) machines before completion. Figure 4.10 depicts the structure of the manufacturing environments.

In this evaluation, we used real problem instances that were extracted from the enterprise resources planning of the manufacturing environment (Krist, 2022, p. 103). The problem instances are described in (Krist, 2022, p. 104). The applied scheduling practices were focused on system efficiency to reduce the total number of major setup times (Krist, 2022, p. 105). As a result, generated solutions may contain delays in delivery dates (Krist, 2022, pp. 105-106). Based on the description of the considered manufacturing environment, we will instantiate the presented artifact and evaluate its performance for solving the discussed problem instances.

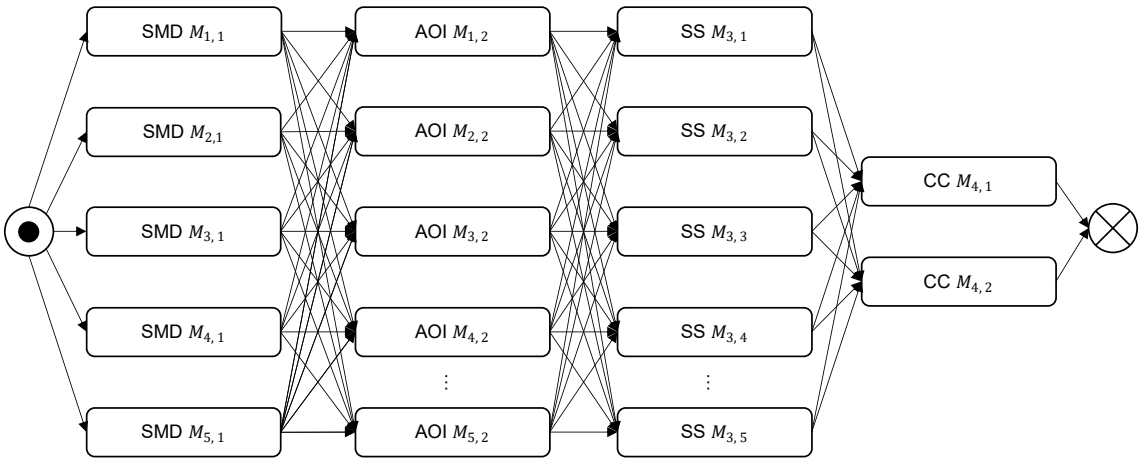


Figure 4.10: The structure of the manufacturing environment (Nahhas et al., 2021b).

4.5.2 Methodology instantiation and problem formulation

In this section, MESEAS is instantiated to address scheduling concerns in manufacturing environments based on the formalization discussed in the previous chapter (cf. Subsection 3.4.3). Equation 4.4 presents the tuple definition of the methodology, which we will discuss in the course of this section. All functionality layers of MESEAS are utilized to address scheduling concerns in this environment and present an overall evaluation of the concept. Namely, the scheduling data model, the simulation, the heuristic library, the optimization, and the machine learning components are therefore instantiated. Given the scheduling problems SP_{multi}, the presented artifact $MESEAS_{(X,\ T)}$ is utilized to find a solution $X \in \mathbb{X}$ for the considered scheduling period T (cf. Section 3.4.2 and Equation 3.1). This solution must minimize the objective values in Γ to address system efficiency and customer satisfaction concerns in manufacturing environments. In the course of this section, we will detail the instantiation of the methodology for several scenarios.

$$MESEAS_{(X,\ T)} = \langle SP_{multi} \mid PI_{multi} \mid \Gamma \mid Sim \mid HL \mid Opt \vee ML \rangle \tag{4.4}$$

The scheduling problem - $\mathbf{SP_{multi}}$

Considering the structural overview of the discussed manufacturing environment presented in Figure 4.10, the scheduling problem can be formulated by instantiating the scheduling data model of the presented methodology. The scheduling data model of MESEAS methodology follows the triple $\langle \alpha \mid \beta \mid \gamma \rangle$ notation presented by Graham et al. (1979) to formalize scheduling problems.

The investigated manufacturing environment consists of four processing stages with heterogeneous machines. The first three processing stages contain five parallel machines that can be utilized to process jobs with different speeds Q_5. The fourth processing stage processes jobs using two identical parallel machines P_2. The structure of the scheduling environment is denoted as Hybrid Flow Shop scheduling environment $\alpha\alpha = HFS_4(Q_5,\ Q_5,\ Q_5,\ P_2)$. The manufacturing operations of the analyzed environment are

characterized by family setup times and eligibility restrictions. The manufacturing operations of the analyzed environment are characterized by family setup times and eligibility restrictions constraints. Jobs are also associated with priority families. The following bullet points briefly summarize the operational constraints and objective measures of the manufacturing environment:

- Family dependent setup times $\beta_1 = f_{g,h}$: Jobs are grouped into families based on their required raw materials. The first and fourth processing stages are subject to family major and minor setup time operational constraints (cf. Section 2.1.2).
- Priority families $\beta_2 = D_{pr}$: Jobs are also classified under several priority families that are associated with the type of customer. This practice is usually to maintain customer relationships and avoid penalties.
- Machine eligibility restrictions $\beta_2 = M_j$: Once the machine starts processing a job $J_j \in J$ of a family f_g in the first stage, it blocks the equipment required for this family. Thus, the rest of the jobs in this family can not be processed by other machines at this stage until the job is completed and the resources are free.
- The makespan $\gamma_1 = C_{max}$: Scheduling practices in the considered environment are oriented toward system efficiency. The makespan was presented in Equation 2.8:
- The total tardiness $\gamma_2 = T$: Delays in delivery dates impact reputation and harm customer relationships. Therefore, scheduling decisions must adhere to the delivery dates of orders. The total tardiness was presented in Equation 2.10.
- The number of major setup times $\gamma_3 = MS$: The configuration time of machines has a significant impact on the efficiency of the considered manufacturing environment and must be minimized. The total number of major setup times was presented in Equation 2.11.
- The total number of penalties $\gamma_4 = U$: To consider customer satisfaction, the minimization of the total penalties is an objective measure of scheduling policies. The total number of penalties was presented in (cf. Equation 2.7).

Based on the description of the considered manufacturing environment, the scheduling problem is expressed by $SP_{multi} = \langle HFS_m \mid f_{g,h},\ D_{pr},\ M_j \mid C_{max},\ T,\ U,\ MS \rangle$.

Problem instance - $\mathbf{PI_{multi}}$

The problem instances include hundreds of customer orders that are divided into monthly backlogs. The customer orders are mapped to the data structure of MESEAS (cf. Section 3.4.2). We further relied on problem instances from related works to investigate the quality of obtained solutions. The following bullet points summarize the instantiation of the scheduling data model:

- The set $T = \{T_t, \ldots, T_{|T|}\} : \forall\ t \in \{1, \ldots, 30\}$ denote a month scheduling period. Given a discrete change in time horizon ΔT_1, we move from T_1 to T_2.
- The set $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, 176\}$ denote jobs in a problem instance, which must be completed during a scheduling period T.
- The $pr_j \in \mathbb{R}^+$ denote a positive integer number associated with every job that indicates the priority of job J_j.
- The set $f = \{f_g, \ldots, f_{|f|}\} : \forall\ g \in \{1, \ldots, 45\}$ denote part-type families of the considered environment. Every family f_g comprises a subset of jobs $J_{j,g} \subset J$.
- The set $D = \{D_{pr}, \ldots, D_{|D|}\} : \forall\ pr \in \{1, \ldots, 20\}$ denote the priority families of the considered scheduling environment. Every family D_{pr} comprises a subset of jobs $J_{j,k} \subset J$. Jobs of a single family share the same priority.
- The set $O_j = \{O_{o,j}, \ldots, O_{|O|,j}\} : \forall\, o \in \{2, \ldots, 4\}$ denote the operations that compose a job $J_j \in J$.
- The set $S = \{S_s, \ldots, S_{|S|}\} : \forall\ s \in \{1, \ldots, 4\}$ denote four processing stages of the considered environment.
- The set $M = \{M_{i,s}, \ldots, M_{m,s}\} : \forall\ i \in \{1, \ldots, 5\}$ denote an m number of machines that are available at every stage $S_s \in S$.
- The $p_{s,i,j} \in \mathbb{R}^+$ denote the required processing time of a job $J_j \in J$ to be completed by a machine $M_i \in M$ on the processing stage $S_s \in S$.
- The $MS \in \mathbb{R}^+$ denotes the overall number of major setup times of all machines M during a scheduling period T. We increase this number subject to the family-dependent setup constraint discussed in Subsection 2.1.2.
- The $ms_i \in [25, 65]$ denote the required time in minutes to setup a machine M_i when switching between two families.

Objective function - Γ

Based on the instantiation of the scheduling data model, the methodology is utilized to explore the solution space $\mathbb{X}$ of the scheduling problem to find a scheduling plan $X \in \mathbb{X}$. This schedule maps the set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$ to the available machines $M = \{M_{i,s}, \ldots, M_{m,s}\} : \forall\ i \in \{1, \ldots, 5\}$ in the corresponding processing stages $S = \{S_s, \ldots, S_{|S|}\} : \forall\ s \in \{1, \ldots, 4\}$. The production schedule $X \in \mathbb{X}$ is composed of a set of matrixes, each detailing a manufacturing plan for a stage S_s such that $X = \{X_s, \ldots,\ X_{|S|}\}\, \forall\ s \in \{1, \ldots, 4\}$.

This production schedule is subject to the discussed constraints. Based on Equation 2.8, Equation 2.10, Equation 2.11, and Equation 2.7, the objective functions are expressed in Equation 4.5. The quality of the investigated schedules during the search is

associated with minimizing the makespan, total tardiness, total number of penalties, and total number of major setup times. Based on the objective functions in Equation 4.5, the overall objective function, which is utilized to assign fitness values to solution individuals, is presented in Equation 4.6.

$$\gamma_1(X) = C_{\max}, \quad \gamma_2(X) = MS, \quad \gamma_3(X) = T, \quad \gamma_4(X) = U \tag{4.5}$$

$$\arg\min_{X \in \mathbb{X}} \Gamma(X) = \arg\min_{X \in \mathbb{X}} [\gamma_i(X), \ldots, \gamma_{|\Gamma|}(X)] : \forall\ i \in \{1, \ldots, 4\} \tag{4.6}$$

Simulation - (Sim)

For solving the scheduling problem, simulation methods are combined with heuristic, improvement, and/or DRL methods. The simulation component was initialized to create simulation models using the Salabim simulation package (cf. Subsection 3.4.4). The optimization and machine learning components then use the simulation models to construct and evaluate solution candidates.

Heuristic Library - (HL)

The simulation model is loosely coupled with the HL component of MESEAS. It initializes the component and invokes the appropriate allocation and sequencing algorithms during the optimization to construct a solution (cf. Subsection 3.4.4 and Section 3.5). Scheduling decisions in the considered environment deal with the allocation and sequencing of jobs in every stage. Therefore, the methodology is instantiated to utilize a set of six allocation algorithms: $A = \{A_a, \ldots, A_{|A|}\} : \forall\ a \in \{1, \ldots, 6\}$. It also utilized a set of six sequencing algorithms: $B = \{B_b, \ldots, B_{|B|}\} : \forall\ b \in \{1, \ldots, 6\}$ to constrcut solutions. The included allocation and sequencing algorithms are:

- A_1 : Family-Increasing Workload-Increasing (FI-WI, Algorithm 7).
- A_2: Family-Decreasing Workload-Increasing (FD-WI, Algorithm 8).
- A_3: Deadline-Aware Family Fit Increasing (DA-FFI, Algorithm 9).
- A_4: Deadline-Workload-Aware Family-Fit-Increasing ($(\text{DWA} - \text{FFI})_{\text{CH}_1}$, based on Algorithm 10).
- A_5: Deadline-Workload-Aware Family-Fit-Increasing ($(\text{DWA} - \text{FFI})_{\text{CH}_2}$, based on Algorithm 10).
- A_6: Deadline-Workload-Aware Family-Fit-Decreasing (DWA-FFD based on Algorithm 10).
- B_1: Earliest Due Date (EDD for single-machine based on Algorithm 1).
- B_2: Shortest Processing Time (SPT for single-machine based on Algorithm 2).

- B_3: EDD-Family Shortest Processing Time (EDD-FSPT, Algorithm 12).
- B_4: EDD-Family Longest-Processing-Time (EDD-FLPT, Algorithm 13).
- B_5: Deadline-Aware Sequencing-Increasing (DA-SI, Algorithm 14).
- B_6: Deadline-Aware Sequencing-Increasing ($(\mathrm{DA}-\mathrm{SI})_{\mathrm{multi}}$ based on Algorithm 14).

Optimization - (Opt)

The optimization component of MESEAS can be instantiated using the presented encoding models in Subsection 3.6.1 and the discussed adopted evolutionary method in Subsection 3.6.2. Based on the business needs, utilizing a single model and a single evolutionary method may be sufficient to address scheduling concerns. For evaluation proposes, we utilized the methodology using all models to systematically investigate their performance and validate their integration with the simulation and heuristic library components. We will elaborate on the instantiation of the presented models in the coming subsections.

Discrete encoding based on family markers (MESEAS-Attribute): Based on the problem formulation, we relied on the family-types attribute to integrate the optimization with the simulation and heuristic library components. Equation 4.7 presents the definition of a solution candidate generated by the optimization model. Based on the conducted analysis of the provided data, the problem instances contain, at most, 46 independent families in the first processing stage. Jobs in the fourth processing stage are grouped into only three families. Every genotype Φ_φ was used to allocate a family f_φ to a machine $M_i \in M$. The solution candidate was decoded using the simulation component, and allocation maps were generated for the family-dependent processing stages. To address the sequencing part of the problem, the DA-SI algorithm was used to sequence jobs in every machine during the simulation. For simplicity and consistent presentation of the computational results, we called this instantiation *MESEAS-Attribute*.

$$\Phi^{attr} = \Big[\Phi_\varphi, \quad \cdots, \quad \Phi_{|f|}\Big] : \quad (\varphi = 1, 2, \ldots, 46) \wedge (\Phi_\varphi = 1, 2, \ldots, 5) \tag{4.7}$$

Discrete encoding based on allocation heuristic markers (MESEAS-Single): The included allocation heuristics discussed in the previous section were used as a marker to define the genotypes for the implemented Non-Dominated Sorting Genetic Algorithms three (NSGA III). Based on the instantiation of the HL, the set $A = \big\{A_a, \ldots, A_{|A|}\big\} : \forall\ a \in \{1, \ldots, 6\}$ denoted a set of six allocation heuristics. The shape of a solution candidate is presented in Equation 4.8. Each genotype Φ_φ initialized an allocation constructive algorithm A_a, which was used at the decision point T_φ. The uncompleted jobs are rescheduled during the simulation of the scheduling period T. For consistency, we utilized the DA-SI algorithm to sequence jobs in every machine (cf. Algorithm 14).

$$\Phi^{A} = \Big[\Phi_\varphi, \quad \cdots, \quad \Phi_{|T|}\Big] : \quad (\varphi = 1, 2, \ldots, 30) \wedge (\Phi_\varphi = 1, 2, \ldots, 6) \tag{4.8}$$

Multi-population discrete encoding based on allocation and sequencing heuristic markers (MESEAS-Multi): The multi-population encoding model was implemented using the allocation and sequencing heuristics as markers. This encoding model also integrated the rest of the sequencing algorithms to be used during the scheduling period. The multi-population encoding model is denoted by the set $\Phi = \{\Phi^A,\ \Phi^B\}$: subject toEquation 4.9.

Similarly, the NSGA III is implemented and initialized to maintain and control the evolution process of the allocation and sequencing populations. It is important to emphasize that both populations coevolve simultaneously during the optimization process. They depend on each other to construct the final solution and obtain the fitness of their independent individuals. Recently, very few similar approaches can be found in the literature that were presented to deal with scheduling problems (Zeiträg et al., 2024; Liu et al., 2023; Wang et al., 2021).

$$\begin{aligned} \Phi^A &= \Big[\Phi_\varphi, \quad \cdots, \quad \Phi_{|T|}\Big] : \quad (\varphi = 1, 2, \ldots, 30) \wedge (\Phi_\varphi = 1, 2, \ldots, 6) \wedge \Phi^A \in \Phi \\ \Phi^B &= \Big[\Phi_\varphi, \quad \cdots, \quad \Phi_{|T|}\Big] : \quad (\varphi = 1, 2, \ldots, 30) \wedge (\Phi_\varphi = 1, 2, \ldots, 6) \wedge \Phi^B \in \Phi \end{aligned} \tag{4.9}$$

Machine Learning - (ML)

The machine learning component was instantiated based on the presented DRL scheduling and evaluation models, which we discussed in Subsection 3.6.3. We adopted the Proximal Policy Optimization (PPO) and the Asynchronous Advantage Actor-Critic (A3C) to address scheduling concerns of the considered manufacturing environment. We can summarize the initialization by mapping our instantiation of the optimization component to the machine learning component using the MDP notation.

Equation 4.10 presents the MDP instantiation of the DRL scheduling and evaluation models using the allocation heuristic encoding. In this encoding, we exposed the DRL agent to the set of allocation heuristics A to select which allocation algorithm must be used at every decision point to schedule jobs. The adopted DRL agent takes an action Φ^A_φ at time T_φ, which is decoded and translated by the initialized HL and simulated by the initialized Sim. Based on the simulation results, the computed reward and the obtained observations are propagated back to the DRL agents. We also supplied the agents with a matrix K^* of key performance indicators for every job. For addressing the sequencing part of the scheduling problem, we similarly relied on the DA-SI algorithm (cf. Algorithm 14).

$$\langle \mathcal{S} = (PI \cup K^* \cup X^* \in \mathbb{X}) \mid \mathcal{A} = HL\ (\Phi^A_\varphi) \mid \mathcal{T} = Sim\ (A_a,\ T_\varphi) \mid \mathcal{R} = -\Gamma_\omega\ \rangle \tag{4.10}$$

For every suggested solution $X^* \in \mathbb{X}$, we computed a set of key performance indicators for the set of jobs $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, n\}$. They are denoted by the set $K = \{K_k, \ldots, K_{|K|}\} : \forall\ k \in \{1, \ldots, |K|\}$. We computed them using the simulation model as presented in Equation 4.11. Each matrix K^* was of the size $(n \times |K|)$ as shown in Equation 4.11. The utilized key performance indicators in the evaluation were the

waiting time (cf. Equation 2.2), the completion time (cf. Equation 2.3), the flow time (cf. Equation 2.4), and the tardiness (cf. Equation 2.6). In essence, the set was instantiated such that $K = \{W_j,\ C_j,\ F_j,\ T_j\}$.

$$K^* = Sim\ (X^*,\ K) = \begin{bmatrix} k_{11} & k_{12} & \cdots & k_{1|K|} \\ k_{21} & k_{22} & \cdots & k_{2|K|} \\ \vdots & \vdots & \ddots & \vdots \\ k_{n1} & k_{n2} & \cdots & k_{n|K|} \end{bmatrix} \tag{4.11}$$

The second encoding exposed the agents to a second controller using the sequencing heuristic to interact with the DRL environment. Equation 4.10 presents the instantiation of the ML component using the allocation and sequencing encodings. The training is conducted in a manner identical to the first encoding to compare the performance of both approaches systematically.

$$\langle \mathcal{S} = (PI \cup K^* \cup X^* \in \mathbb{X}) \mid \mathcal{A} = HL(\Phi^A_\varphi,\ \Phi^B_\varphi) \mid \mathcal{T} = Sim(A_a,\ B_b,\ T_\varphi) \mid \mathcal{R} = -\Gamma_\omega \rangle \tag{4.12}$$

The training of the adopted A3C and PPO using the instantiated DRL scheduling and evaluation model was conducted using the same reward function $-\Gamma_\omega$. DRL agents take action to maximize their reward value. Therefore, we transformed the minimization objective function to maximization weighted reward function $-\Gamma_\omega$ as depicted in Equation 4.13. The penalty function was multiplied by ten to incentivize agents to avoid penalties and violations in delivery dates. In the discussed DRL approaches, we relied on multi-discrete action space. Multi-discrete action allows a DRL agent to take multiple actions at the same time and is usually adopted to train agents in complex environments (Delalleau et al., 2019; Kanervisto et al., 2020). After the agent suggests multiple discrete actions, the simulation model sequentially processes them to return the immediate reward.

$$\arg\max_{X \in \mathbb{X}} \Gamma_\omega(X) = -\ (\gamma_1(X) + \gamma_2(X) + \gamma_3(X) + 10 * \gamma_4(X)) \tag{4.13}$$

4.5.3 Computational results using optimization component

Component configuration

We conducted experiments that utilized all functionality layers of the presented artifact with emphasis on the full integration of the presented components. Therefore, the presented analysis focuses on the instantiation of the multi-population configuration of the optimization component. We finally compare the obtained results with other methods from related works in Subsection 4.5.5. The optimization component was initialized using the NSGA III to solve the scheduling problems subject to multiple objective values presented in Equation 4.6. The concept of reference points was adopted by Deb and Jain (2014, p. 581) to guide the optimization in searching the objective space and ensuring diversity. We relied on the uniform reference points that were adopted by Deb and Jain

(2014, p. 581) based on the investigation of Das and Dennis (1998, p. 642) to parameterize the NSGA III with $P = 4$. This parameterization results in $\psi = 35$ reference points (Deb and Jain, 2014, p. 581 ;Das and Dennis, 1998, p. 642).

We adopted the two-point crossover operator with $\sigma = 0.8$ crossover probability to conduct the evaluation. The two-point crossover evolutionary operator is often adopted in the design of evolutionary methods due to its simple design. It has a rich theoretical background and proved to be efficient empirically (Holland, 1975; de Jong, 1975, pp. 20-21). A shuffling index, introduced by Eshelman et al. (1989, p. 15), mutation function was utilized with a probably $\delta = 0.7$. The population size and the number of individuals were set with $\mu = \lambda = 250$, respectively. However, a larger population size reduces the stochastic behavior of an evolutionary method. It may yield better performance in the long run at the expense of low quality at the early stage of the optimization (de Jong, 1975, pp. 191-192). We finally set the max number of generations to $maxGen = 200$. We mentioned earlier that the parameters of the optimization component were obtained empirically with emphasis on efficiency subsubsection 4.3.3.3. Evolutionary optimization methods are eventually stochastic optimization techniques. Therefore, we conducted 50 optimization experiments to solve every problem instance, amounting to 1500 experiments. The objective is to verify the integration of the optimization component with the simulation and heuristic library components. The used parameters are summarized in Table 4.3.

Table 4.3: Evolutionary operators and parameterization.

Operator	**Utilized**	**Parameterization**
Initialization	Random population	Population size $\mu = 250$
Selection	NSGA III	Individuals $\lambda = 250$
Reference points	Uniform normalized	$P = 4$, $\psi = 35$
Recombination	Uniform two-point crossover function	$\sigma = 0.8$
Mutation	Shuffling index function	$\delta = 0.7$

Analysis of computational results

The experiment's results for minimizing the makespan are presented in Figure 4.11. The upper part of the figure, Makespan (A), depicts the results of the optimization component for solving the first fifteen problem instances. The lower part of the figure, Makespan (B), demonstrates the collected computational results for solving the second fifteen problem instances. Every point in both parts of the graph represents the makespan of an investigated solution during the optimization. The makespan data points represent the makespan from the Pareto fronts of fifty optimization experiments per problem instance.

The boxplots depict the makespan of Pareto fronts solutions for every problem instance using the Interquartile Range (IQR) . According to the IQR, every boxplot is divided into four parts (quartiles), excluding outliers. Outlier data points are plotted outside the IQR range, as shown in the figure. The first quartile includes the values of the makespan that are 25% greater than all values and 75% less than all found makespan in

the Pareto fronts. It marks the IQR found minimum, excluding outliers. For instance, in the first problem instance, the best-found solution for minimizing the makespan over the fifty optimization experiments was 16,386. Also, 75 % of explored solutions are below 16805. The second quartile divides the dataset of the best makespans into two parts and is located in the middle. For instance, the mean of the makespan for solving the first problem instance was 17,122.1, and the median was 17058.

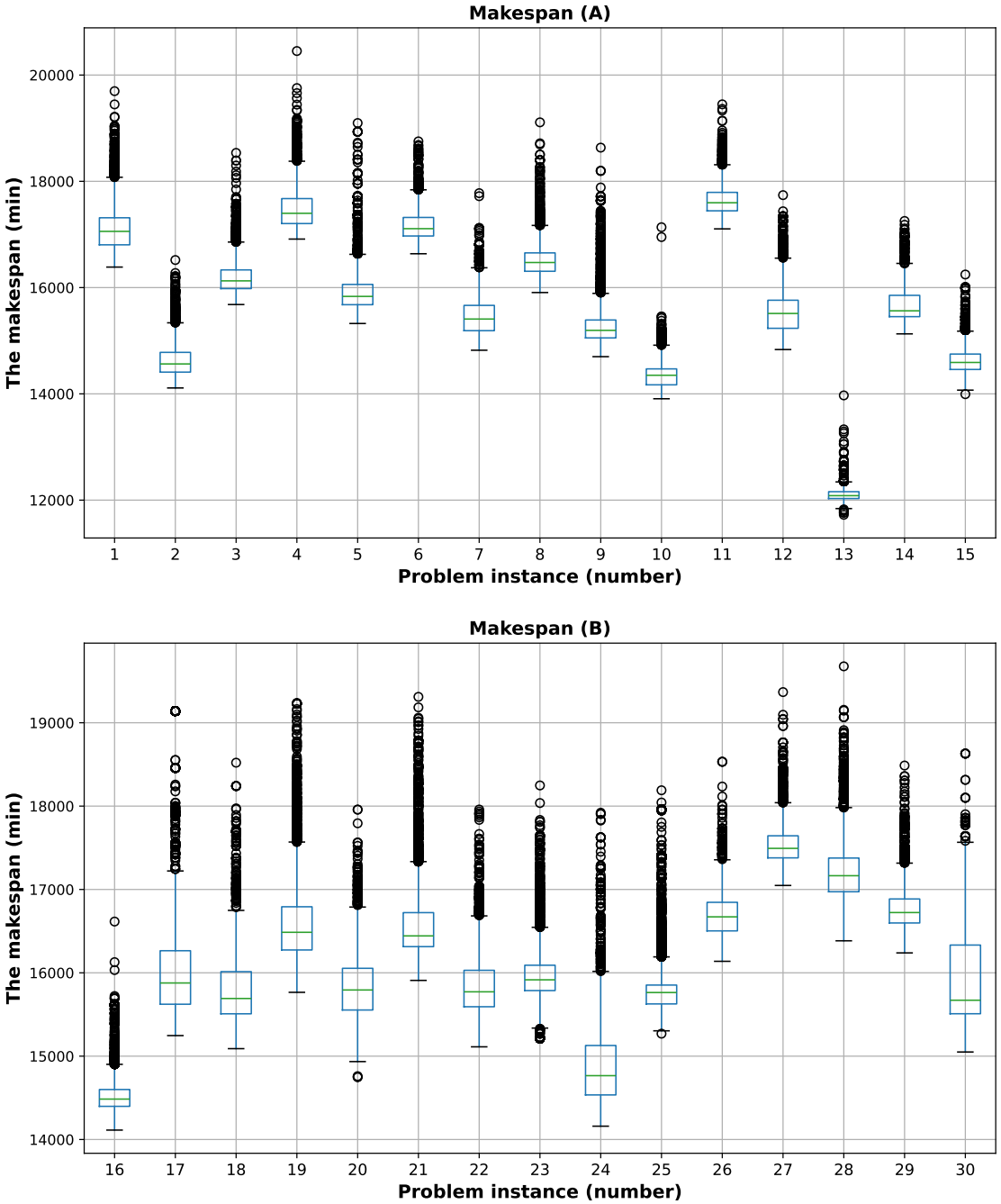


Figure 4.11: Computional results for minimizing the makespan.

The third quartile is opposite to the first one and includes makespan values that are 25 % less than all values and 75 % greater than all values. The fourth quartile encompasses the rest of the makespan values until the IQR max. For instance, the worst-found IQR makespan in solving the first problem instance was 18,097.8. The actual worst makespan, including outliers, was 19,797 minutes.

It is noticeable that for several problem instances, such as 9, 21, and 25, many outliers were present in the Pareto fronts of the fifty optimization experiments. That implies that the solution space was explored extensively and that the investigated objectives are conflicting. Their business background can explain the conflicting nature of the objective values. The makespan and the total tardiness are system efficiency business objectives, while the total tardiness and the total number of penalties are customer satisfaction business objectives.

In summary, the results of the experiments delivered solutions with an average minimum makespan amounting to 15,306 for solving problem instances. The average of the best 25 % of all found makespan values for solving all problem instances was 15,685.44 with a 2.35 % deviation from the minimum average. The average of the best 75 % of all found makespan values for solving all problem instances was 16,087.44 with a 4.855 % from the average minimum. In conclusion, 50 % of best-found solutions for solving all problems deviated on average by 2.62 % from the average minimum. The median of all experiments for solving all problems was 15,852.6.

The experiment's results for minimizing the total major setup times are demonstrated in Figure 4.12. Similarly, the upper part of the figure, Total major setup times (A), depicts the results of the optimization component in minimizing the total major setup times for solving fifteen problem instances. The lower part of the figure, Total major setup times (B), demonstrates the computational results of the rest of the considered problem instances. The results depicted in the figure were collected from the Pareto fronts of the same experiments presented per problem instance. Each boxplot summarizes statistical findings for solving a problem instance.

The average optimal lower bound of the major setup times for solving the problem instances is 34. This lower bound was calculated by averaging the total number of families in problem instances. It implies that once a machine starts processing a family, it must process all the jobs within the family with no regard to any other objective value. The collected optimization results from the optimization component showed that the average minimum of 38 deviates by 10.5 % from the possible global minimum for solving the problem instances. The minimization of the other objective values obviously explains this deviation. The average of the best 25 % of all found major setup values for solving all problem instances was 41,3, with a 7.9 % deviation from the minimum average. The average of the best 75 % of all found solutions was 44,5, with a 7.1 % deviation from the average minimum. In conclusion, the median of all experiments for solving all problems to minimize the total major setup times was 42.7.

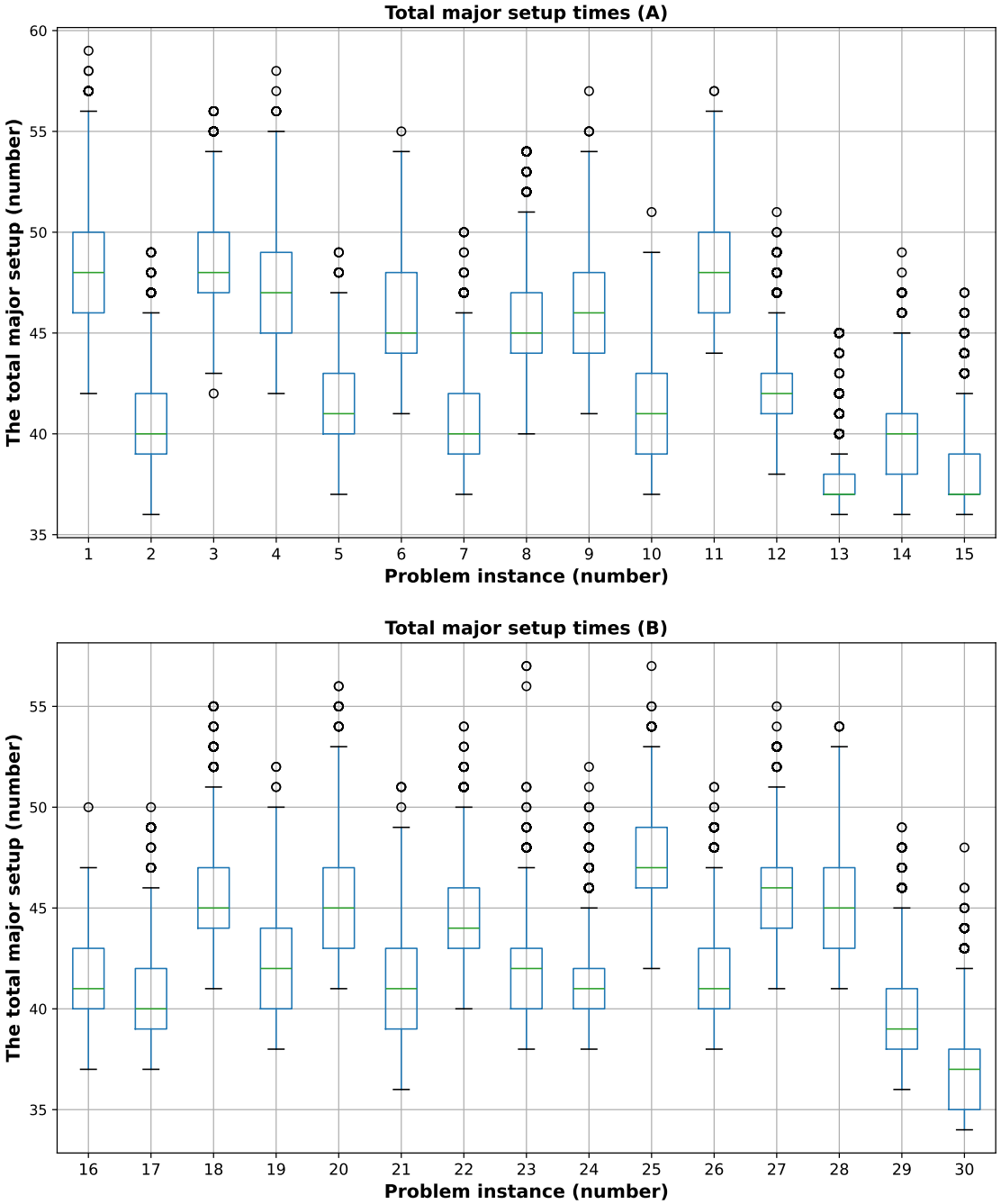


Figure 4.12: Computional results for minimizing the total number of major setup times.

The experiment's results for minimizing the total tardiness are presented in Figure 4.13. The upper part of the figure, Total tardiness (A), presents the results of the optimization component for solving the first fifteen problem instances. The lower part of the figure, Total tardiness (B), displays the computational results for solving the second fifteen problem instances. Similar to the previous graph, every point in both parts of the figure represents the total tardiness of a solution. The total tardiness data points in the figure were collected from the Pareto fronts of fifty optimization experiments per problem instance.

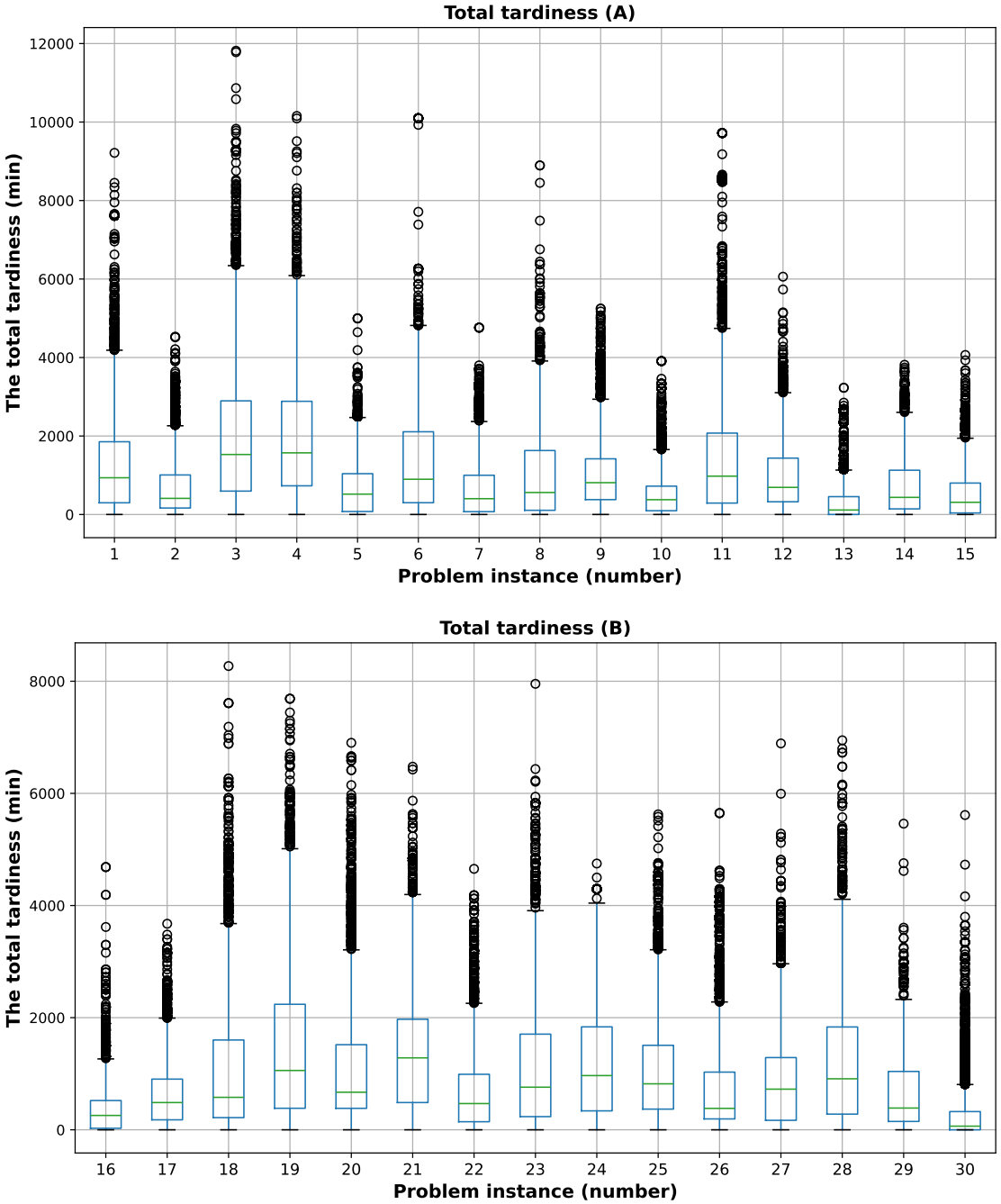


Figure 4.13: Computional results for minimizing the total tardiness.

For instance, the best % 25 of collected best solutions from fifty optimization experiments had 298.3 minutes of total tardiness. However, to solve the same problem, 200 unique solutions with zero total tardiness were found. Obviously, the minimum average overall optimization runs is zero. It is evident that in solving all problem instances, many outliers were present in the Pareto fronts of the optimization experiments. That points to a proper exploration of optimization to investigate solutions in different regions. The results of the total tardiness align with the makespan one, revealing the conflicting nature of the objective values. The optimization component delivered solutions minimizing the

total tardiness to zero for solving all problem instances, as displayed in the figure. Hence, the average minimum of the total tardiness for solving all problem instances was zero. The average of the best 25 % of all found total tardiness values for solving all problem instances was 238.9 minutes. The average of the best 75 % of all found solutions had roughly under a day of total tardiness.

In summary, the median total tardiness for solving all problem instances was 678.73. The total tardiness and the number of penalties objective values have a strong correlation. Both business objectives significantly conflict with minimizing the total number of major setup times and somewhat conflict with minimizing the makespan.

The experiment's results for minimizing the total number of penalties over a scheduling period are presented in Figure 4.14 and Figure 4.15. Total penalties (A), depicts the results of the optimization component in minimizing the total penalties for solving the first fifteen problem instances. Total penalties (B), demonstrates the computational results for solving the second fifteen problem instances. The total penalty data points in the figure were collected from the Pareto fronts of the same experiments and are presented per problem instance. Every boxplot demonstrates the statistical analysis for solving a problem instance. The presence of the outliers in solving all problem instances explains the discussed outliers in the analysis of the total tardiness results. For instance, for solving the first problem instance, the average minimum is zero, and the median is three penalties.

Nevertheless, 200 unique solutions with zero penalties were collected from the experiments conducted to solve the problem. The statistical analysis of the collected solutions for solving all problem instances showed that the best 25 % of solutions contained, on average, a 1.33 penalty. The median of the total number of penalties for solving all problem instances was 2.57.

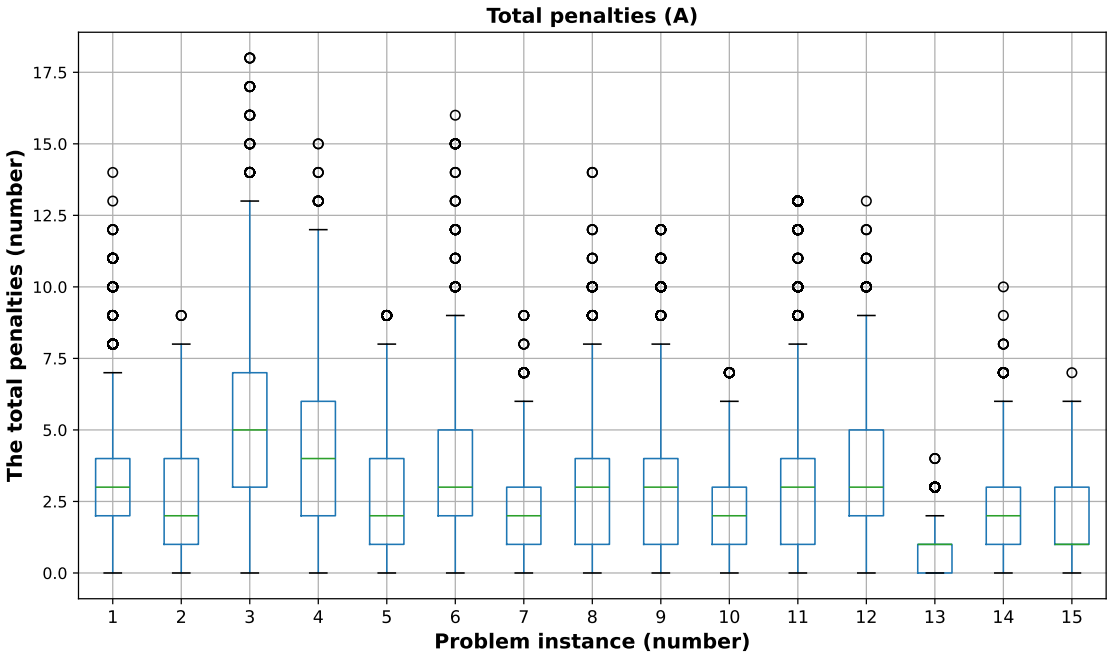


Figure 4.14: Computional results for minimizing the total penalties (A).

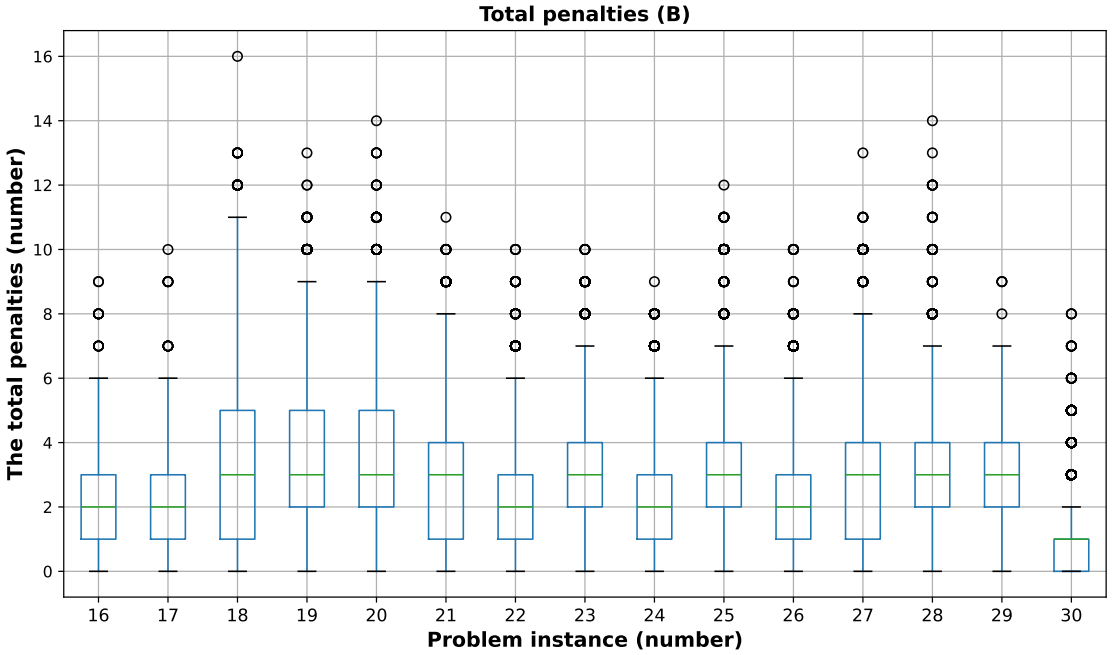


Figure 4.15: Computional results for minimizing the total penalties (B).

Detailed analysis of obtained results

The findings of the statistical analysis of the conducted experiments highlighted the conflicting nature of the objective values. The performance of the considered scheduling environment is also influenced by the total number of major setup times in the fourth processing stage. We excluded the minimization of this objective value in the statistical analysis to sustain comparability to related works. Nonetheless, we evaluated the methodology's performance for minimizing five objective values, including the major setup times in the last stage.

For instance, Figure 4.16 depicts the investigated solutions for solving the first problem instance. The X-axis and the Y-axis demonstrate, in minutes, the makespan and the average tardiness, respectively. For clarity, we accumulated the number of major setup times in the first and fourth stages and used the Z-axis to represent their value. The depicted data points in green represent investigated solution individuals during the optimization. They are distributed in the three-dimensional figure based on objective values. The small red triangles are a set of best solutions from the Pareto fronts of the conducted optimization.

Since the problem is formulated as a minimization optimization problem, the best solution instances are, to a large extent, concentrated in the lower part of the figure. Here, we want to point out that objective values were normalized before assigning fitness values to the investigated solution individual. In this presentation, we wanted to elaborate on the business impact of the considered objective values. As shown in the figure, attempting to minimize the total major setup times has a significant impact on the total tardiness and a noticeable influence on the makespan.

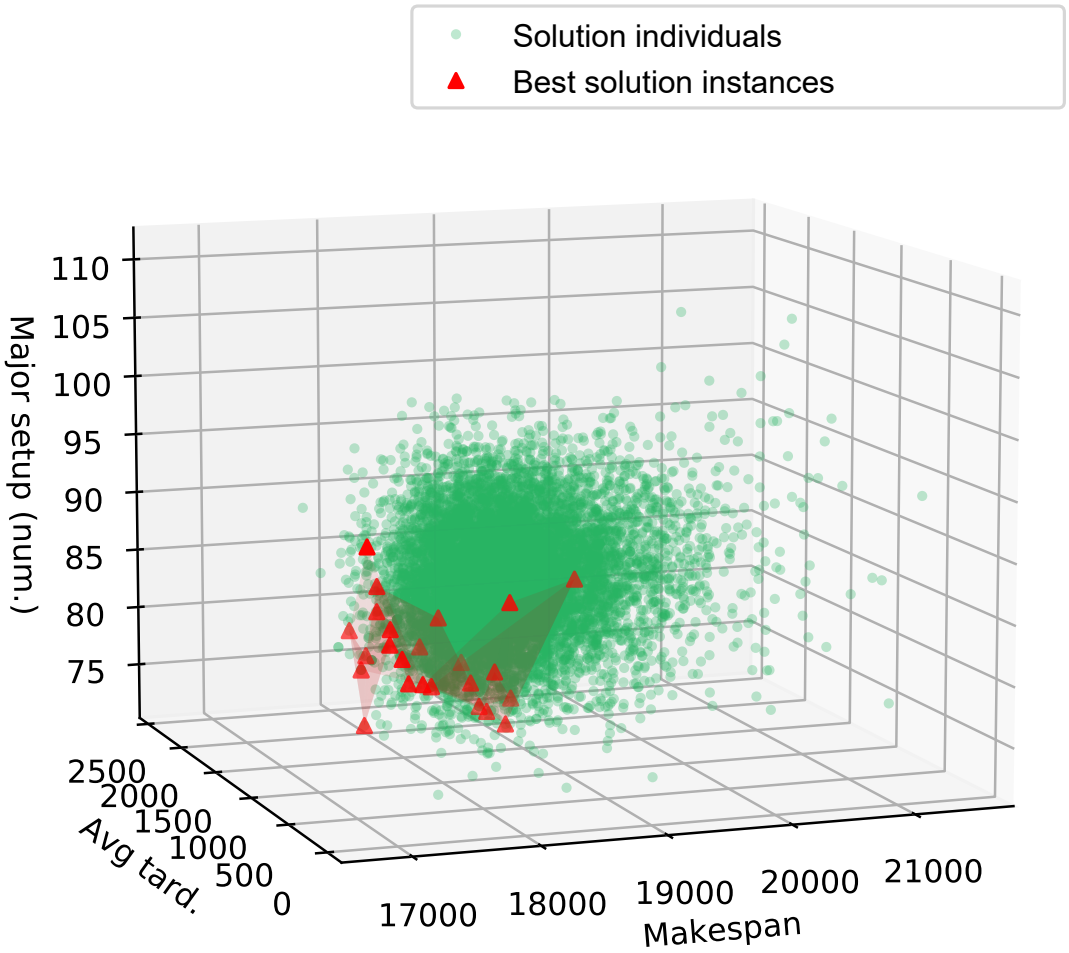
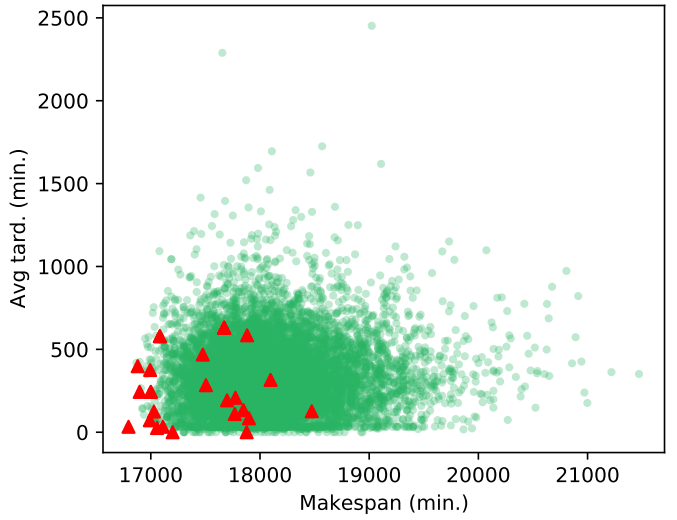
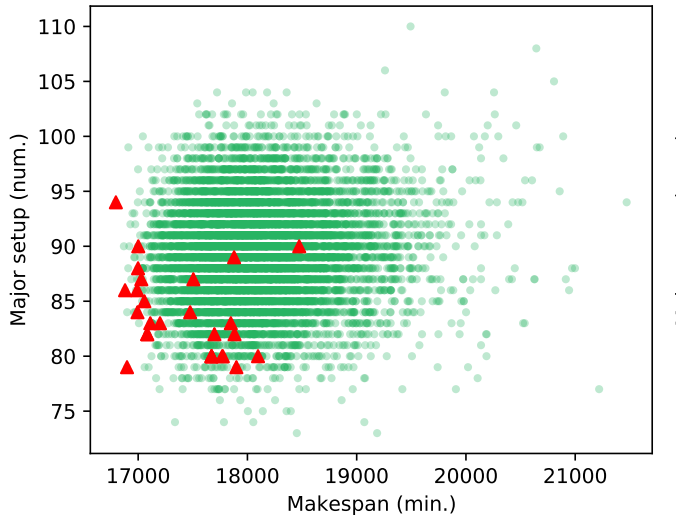


Figure 4.16: Example representation of explored solution candidates and their objective values for solving the first problem instance (Nahhas et al., 2022a).

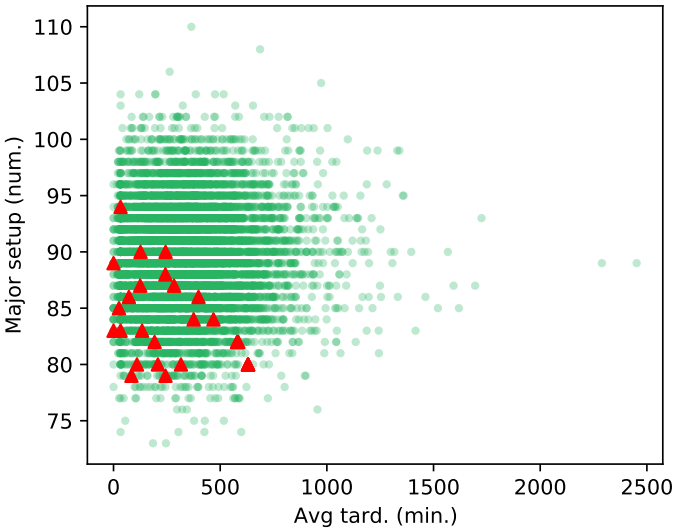
The results also showed that the presented methodology sufficiently explores the solution space and maintains diversity in the population during the optimization despite considering five objective values. We observed in the backside of the figure that a large number of solution individuals yielded significant improvement in minimizing the total number of major setup times while sustaining a very high makespan. Figure 4.17 depicts the computational results for solving the problem using two-dimensional representation. The upper part of the figure displays the quality of the best solutions in terms of minimizing the makespan and the average tardiness. The average tardiness was used here to include the total tardiness and the total number of penalties. As shown in the figure, a set of best-found solutions is concentrated in the lower-left part of the figure, leaning toward the minimal makespan and the minimal average tardiness. The second set of the best-found solutions is located to the right.



(a)



(b)



(c)

Figure 4.17: Example representation of explored solution candidates -conflicting nature of objective values in solving the first problem instance (Nahhas et al., 2022a).

The central part of the figure compares the makespan against the accumulated major setup times in the first and fourth processing stages (cf. Figure 4.17). The results showed two sets of best-found solutions, designated in red. Comparing the presentation of the results in the central part of the figure with the upper part confirmed the conflicting nature of the makespan and the major setup times objectives. In the lower-left part of the figure, the best solutions that minimized the makespan and the total setup times on both stages were concentrated. The second set of best solutions was somewhat distributed around makespan values amounting to roughly 17,900. This set of solutions delivers a slightly better accumulated major setup, obviously, at the expense of the makespan.

Finally, the lower part of the figure displays the accumulated major setup times and the average tardiness of collected solutions during the optimization. It is also evident that the two sets of best solutions were identified. The first set is also concentrated in the lower left part of the figure, minimizing the average tardiness and the major setup times. In summary, integrating the optimization component with the rest of the components in MESEAS showed robust performance even when we considered further objective values in the optimization. In fact, the lowest makespan was found after considering the fifth objective value: 16,661 minutes compared to 17,308 minutes, which we discussed in the previous analysis. We selected solutions that also achieved zero total tardiness and penalties.

Analysis from business perspective

The presented analysis builds up on and summarizes the evaluation results of the optimization and machine learning components published, for instance, in (Nahhas et al., 2021b, Nahhas et al., 2022a, Nahhas et al., 2022b, Nahhas et al., 2024a). The first adoption of GA using attribute markers is often found in related works for solving scheduling problems. We combined GA with simulation and heuristic techniques to address the allocation and sequencing part of the scheduling problem. The GA is adopted mainly to solve the allocation part of the problem. To address the sequencing part of the problem, we utilized one of the algorithms that had been developed in the heuristic library, namely, the Deadline-Aware Family-Increasing (cf. Algorithm 14). We will call this combination (GA & DA-FI). The multi-population setup of MESEAS that integrates the full functionalities of all components except the ML component will be abbreviated by (MESEAS-Multi).

Table 4.4 presents selected solutions that yielded the best results to optimize the objective values in solving the discussed thirty problem instances. Starting with the makespan, computational results of MESEAS-Multi demonstrated an average of 15,926.83 minutes makespan compared to 17,957.40 minutes using the (GA & DA-FI). This difference amounted to a 10.97 % better performance in minimizing the makespan, which has a significant impact on the overall system efficiency. The average of absolute found minimum makespans by MESEAS-Multi for solving the thirty problem instances was 15.2 %. The absolute minimum of solution, given the multi-objective nature of the problem, implies that we disregard the importance of all other objective values.

Table 4.4: Evaluation of MESEAS for solving thirty problem instances in a multi-stage scheduling environment modified based on Nahhas et al., (2021b).

	MESEAS - Multi				**Sim (GA & DA-SI)**				**Sim (EDD)**			
PI	$\mathbf{C_{max}}$	**MS**	**T**	**U**	$\mathbf{C_{max}}$	**MS**	**T**	**U**	$\mathbf{C_{max}}$	**MS**	**T**	**U**
1	17,308	46	0	0	19,056	52	0	0	19,685	132	121	1
2	14,111	40	0	0	15,691	42	0	0	16,339	134	790	1
3	16,482	48	0	0	16,886	50	0	0	18,766	127	0	0
4	18,066	51	0	0	19,281	56	0	0	19,890	130	0	0
5	15,892	39	0	0	17,928	44	0	0	18,694	121	49	1
6	17,319	43	0	0	17,648	50	0	0	20,182	127	605	3
7	16,083	38	0	0	16,834	43	0	0	17,602	129	0	0
8	16,984	42	0	0	20,467	46	0	0	18,124	135	0	0
9	15,375	43	0	0	15,658	46	0	0	16,703	134	0	0
10	14,252	39	0	0	15,936	40	0	0	16,876	132	0	0
11	17,737	46	0	0	20,522	57	500	2	19,546	134	1,499	1
12	15,219	43	0	0	19,232	43	0	0	17,596	129	0	0
13	12,060	37	0	0	14,606	39	0	0	15,438	133	0	0
14	15,489	41	0	0	17,605	42	0	0	17,899	128	0	0
15	14,434	37	0	0	16,691	41	0	0	16,224	134	0	0
16	14,648	38	0	0	15,918	46	0	0	16,699	139	0	0
17	15,589	40	0	0	18,886	43	0	0	18,817	135	0	0
18	16,099	43	0	0	19,767	46	0	0	17,994	128	0	0
19	16,532	41	0	0	17,921	50	0	0	19,371	132	0	0
20	15,845	44	0	0	18,083	44	0	0	18,147	135	0	0
21	16,434	40	0	0	17,480	48	0	0	19,500	126	0	0
22	15,640	44	0	0	15,771	46	0	0	17,698	138	407	1
23	15,671	42	0	0	19,838	45	0	0	18,450	127	0	0
24	14,529	41	0	0	16,440	43	0	0	17,536	138	0	0
25	15,574	48	0	0	19,190	52	0	0	17,104	142	404	1
26	16,851	39	0	0	21,300	48	0	0	18,210	125	0	0
27	17,599	43	0	0	21,190	48	0	0	19,031	137	0	0
28	17,361	43	0	0	17,779	51	0	0	20,314	140	0	0
29	16,873	40	0	0	18,225	46	0	0	19,545	131	115	1
30	15,749	35	0	0	16,893	38	0	0	19,378	130	0	0

Compared to the simulation-based EDD, MESEAS-Multi delivered solutions with a 12.72 % better makespan on average. To summarize the analysis of the makespan, the best-found solutions using MESEAS-Multi deviate on average from the absolute found minimum by only 2.3 %. To minimize the total major setup times, MESEAS-Multi also dominated the conventional heuristic and genetic algorithms for solving the thirty problem instances. On average, MESEAS-Multi's best scheduling solutions are 9.2 % superior to the ones found by the combination of GA and DA-SI. The simulation-based EDD best solutions contained a significantly high number of major setup times, amounting to an average of 132 overall problem instances. Statistically, MESEAS-Multi's best solutions minimized, on average, the total of major setup times by 68.32 % compared to simulation-based EDD. This minimization amounted to a 218 % improvement in business objective value. Priority rules suffer on performance if solving a scheduling problem is subject to minimizing multiple objective values. Both (GA & DA-FI) and EDD failed to find solutions that eliminated penalties and total tardiness. In fact, the combination of EDD and simulation delivered solutions with penalties and total tardiness for solving eight problem instances, which correspond to 26.7 %.

Since the majority of best solutions using MESEAS-Multi and (GA & DA-SI) minimized the total tardiness and penalties to zeros, we will summarize the analysis of the computational results with an emphasis on system efficiency. Figure 4.18 depicts the overall analysis of best solutions for solving the problem instance in terms of minimizing the makespan. Figure 4.19 summarizes the overall analysis of best solutions in terms of minimizing the total major setup times. In both figures, the best scheduling solutions of MESEAS-Multi are presented using blue bars. The best GA scheduling solutions are presented using red bars. The gray bars demonstrate the objective values of constructed scheduling solutions using the simulation component combined with the EDD scheduling rule. The green line with markers summarizes the comparison between the MESEAS and conventional GA integrated into MESEAS in percentages from a business perspective. The horizontal axis depicts the unique identifiers of the considered problem instances. We used the primary vertical axis (left side) to represent the objective values of solutions while using the secondary vertical axis (right side) to demonstrate the performance difference between MESEAS and GA. For clarity, we sorted the raw results in an ascending order by the difference in performance. As shown in Figure 4.18, MESEAS delivered better solutions by at least 0.8 % for solving problem 22 and reached 21.0 % improvement for solving problem 23 with an average of 10.97 % superior performance. As demonstrated in Figure 4.19, MESEAS and GA found solutions with identical quality in minimizing the major setup times for solving two problem instances. Then, MESEAS outperformance of GA started at 2.4 % in solving problem 14 and increased the margin to 19.3 % for solving problem 11 with an average of 9.17 %.

In summary, the evaluation showed that integrating simulation, heuristic, and meta-heuristic for addressing multi-stage scheduling problems yielded significant improvement in solving scheduling problems. The comparison to related works and used scheduling practices in the considered environment will be summarized in Subsection 4.5.5.

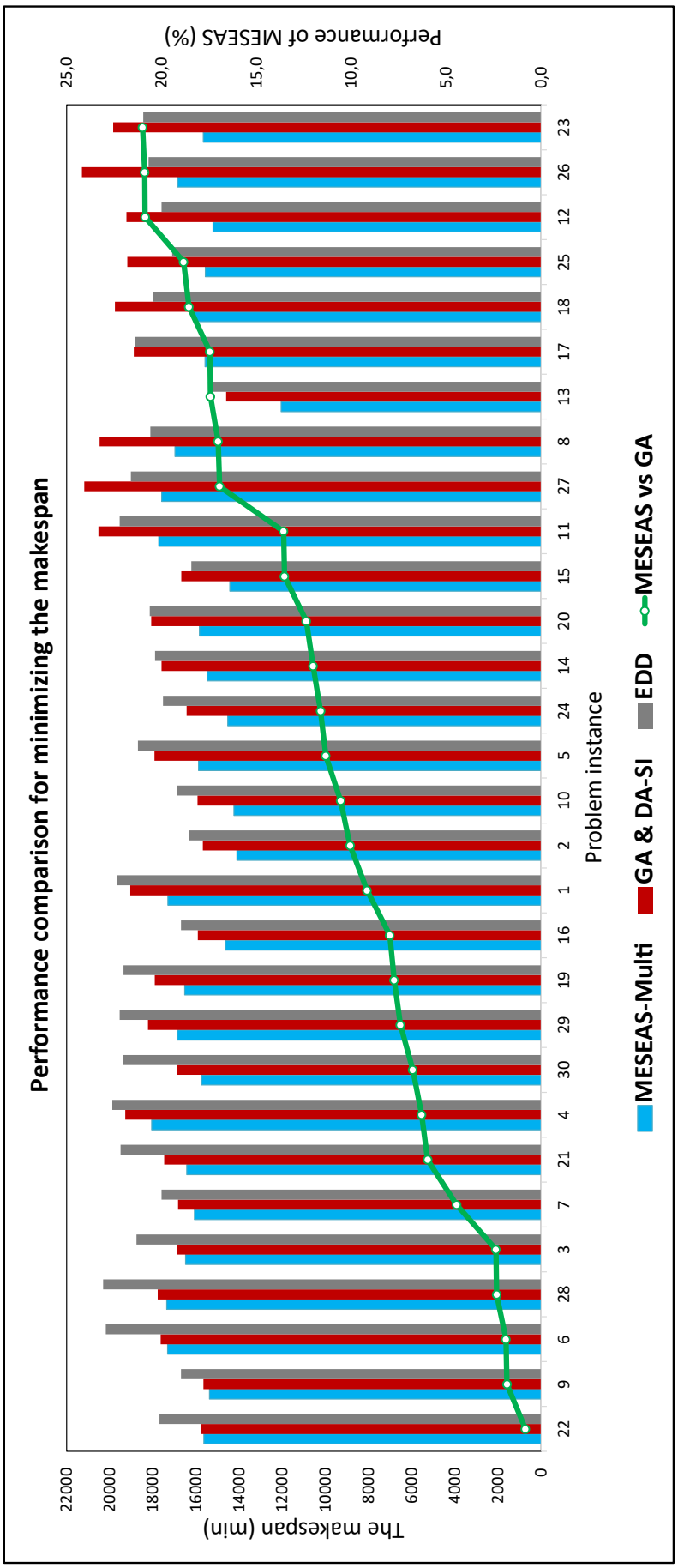


Figure 4.18: Performance comparison in terms of makespan in percent (Nahhas et al., 2022b).

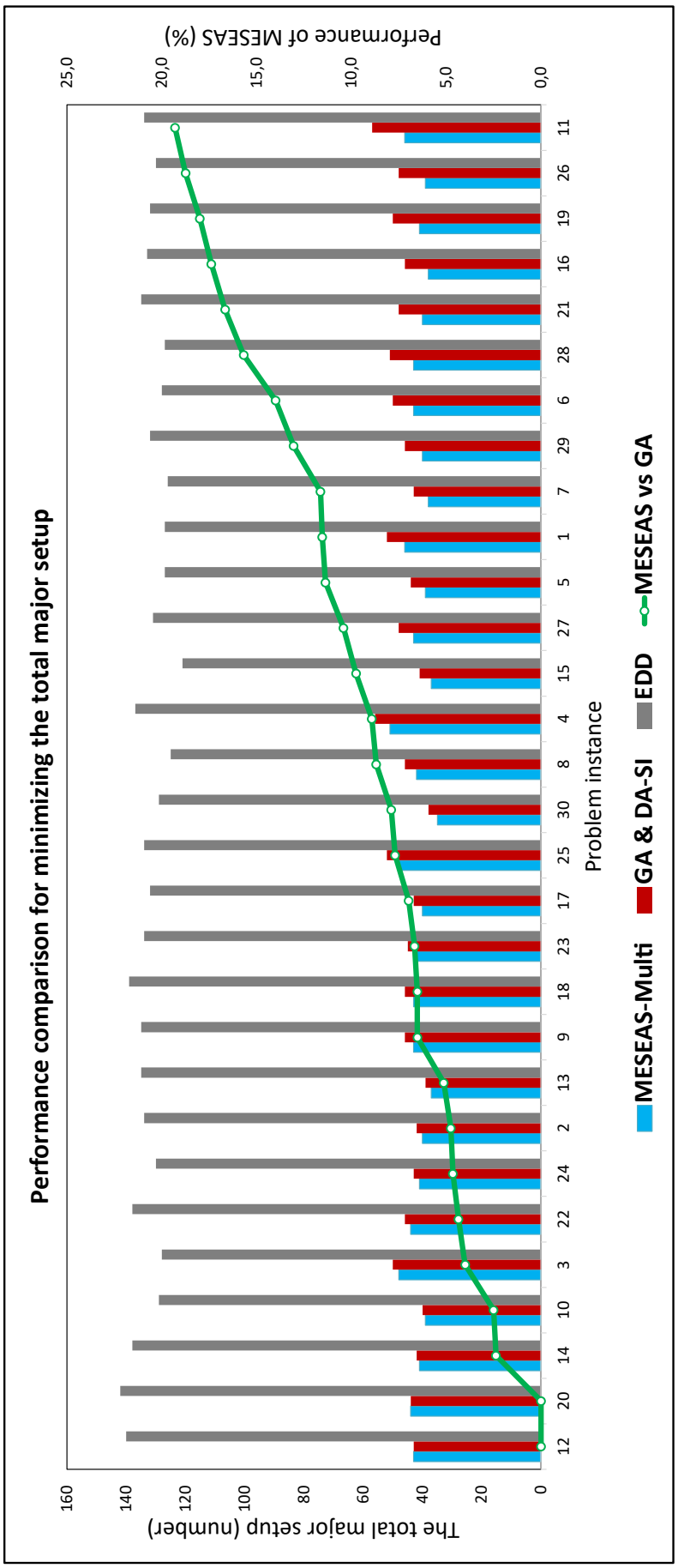


Figure 4.19: Performance comparison in terms of major setup times in percent (Nahhas et al., 2022b).

4.5.4 Computational results using machine learning component

Component configuration

Based on the instantiation of MESEAS for solving multi-stage scheduling problems, the component was initialized using the adopted A3C and PPO DRL methods. Both algorithms can be utilized in the presented artifact to deal with scheduling concerns. For evaluation, we conducted independent DRL experiments on the A3C and PPO algorithms using different formulations of the action space (cf. Section 4.5.2). We relied on a weighted sum approach to include the considered objective values. We instantiated the reward function used for training as presented Equation 4.13. The presented experiments emphasize the integration of all functionality layers of the presented artifact. Both DRL methods utilized the presented scheduling data model, the simulation component, the heuristic library component, and the encoding models. Based on the formulated actions space, each DRL method is trained in an independent experiment to select among allocation and sequencing heuristics over the scheduling period to solve the scheduling problems (cf. Section 4.5.2). To establish confidence in the performance of the methods, we evaluated them to solve problems from related works (cf. Subsection 4.5.5).

The hyperparameters of the DRL algorithms were selected based on preliminary tuning and investigation of the suggested default configuration of the algorithms. All adopted DRL methods in the presented artifact utilized fully connected Deep Neural Networks (DNN). The main parameters of DNN are the number of hidden layers, the number of neurons in each layer, and the type of activation function Koutsoukas et al. (2017). All algorithms are trained using four hidden layers with this network configuration $\mathrm{DNN}_{\mathrm{size}} = [2048, 2048, 1024, 1024]$. Every element in the vector denotes the number of neurons in the corresponding hidden layer. We utilized the ReLU activation function to train both algorithms to maintain comparability.

We configured the learning rate by every algorithm to $\varepsilon = 0.0001$. Finding the right learning value is critical since a value that is too small can make the training process much slower. On the other hand, a learning rate that is too high might prevent agents from exploring new possible solutions. This behavior can result in either low accuracy or an inability to explore better solutions (Wu et al., 2019). We configured the entropy coefficient hyperparameter to $\varepsilon = 0.4$. The entropy parameter controls balancing the exploration and exploitation of agent behaviors during training (Ahmed et al., 2018). Finally, the discount factor parameter is set to $\gamma = 0.05$. Sometimes, the MDP formulation is extended to include the discount factor $\gamma \in [0,\ 1]$. This parameter modulates the overall strategic conduct of a DRL algorithm. Parameterizing γ with a value closer to zero motivates a DRL algorithm to maximize its immediate reward at each step. That is ideal for our formulated multi-discrete action space. Selecting a value closer to 1 compels the DRL agent instead to strive for higher cumulative rewards. The critic's networks in A3C and PPO are activated during the training in all experiments. A summary of the most essential hyperparameters is presented in Table 4.5.

Table 4.5: Summary of selected hyperparameters of adopted algorithms.

Algorithm	**Parameter**	**Parameter value**
A3C	Learning rate	$lr = 0.0001$
	Discount factor	$\gamma = 0.05$
	Entropy coefficient	$\varepsilon = 0.4$
	Training batch size	batch $_{\text{size}} = 400$
	Activation function	ReLU
PPO	Learning rate	$lr = 0.0001$
	Discount factor	$\gamma = 0.05$
	Entropy coefficient	$\varepsilon = 0.4$
	Training batch size	batch $_{\text{size}} = 100$
	Activation function	ReLU

Analysis of computational results

Every adopted algorithm was evaluated by launching ten independent training experiments since both techniques contain stochastic elements that may affect their performance. In this training, the DRL methods decided which allocation algorithm was used at every decision point during the scheduling period. The DA-SI sequencing algorithm was utilized to sequence jobs on the machines in the first and fourth processing stages with family setup times. Then, the EDD was used to sequence jobs on machines in the second and third processing stages. Both DRL methods were trained on three problem instances of distinct complexity. We trained agents to deal with problem instances with medium complexity. Afterward, agents were exposed during the training to a simpler problem instance to observe loss of performance. Finally, the most complicated problem instance concluded the training of the agents. The complexity of the utilized problem instances was empirically recognized using the optimization component. It strongly correlates with the number of major setup times and the makespan (cf. Table 4.6). The objectives were to investigate the performance of the adopted DRL methods in dealing with scheduling problems of different natures and investigate their ability to adapt.

The collected computational results of the experiments are presented in Figure 4.20. The performance of the algorithms is compared in terms of their ability to maximize their reward value per episode during the optimization. The vertical axis in the figure is utilized to depict the mean episode reward over ten experiments. The performance of the A3C algorithm for solving the problem instances is depicted in blue. The orange line displays the progress of the PPO in solving the considered problems. The agent starts interacting with the simulation environment to solve the first problem instances for 25,000 steps. As shown in the figure, the A3C algorithm started to find the correct trajectory after 6,000 steps and demonstrated a steady learning curve in dealing with the first problem instance.

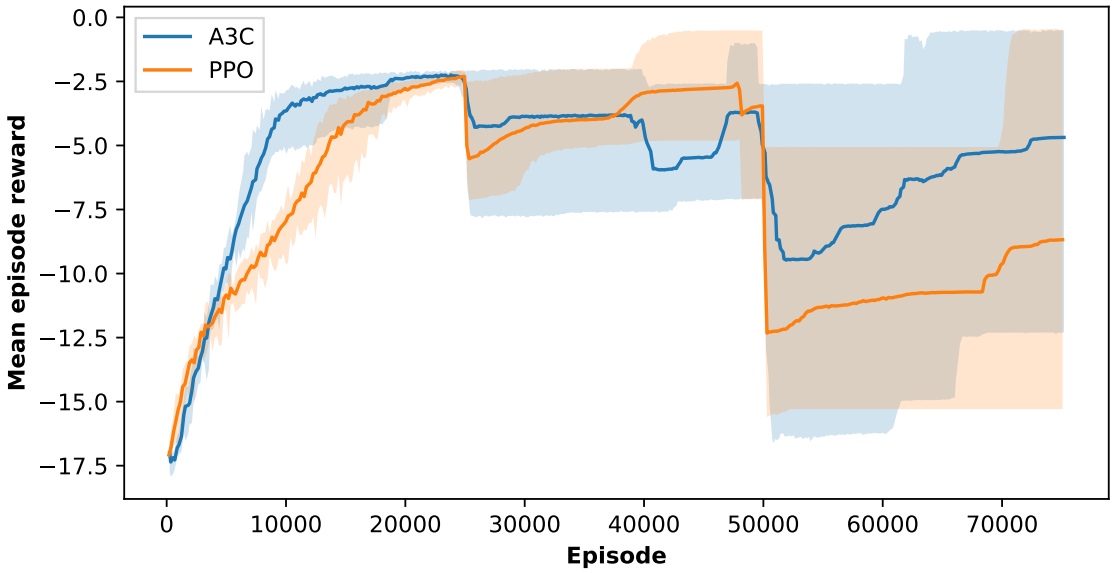


Figure 4.20: Computational results on the performance of MESEAS-A3C and MESEAS-PPO using the allocation approach (Nahhas et al., 2022b).

The performance of the PPO similarly showed a steady increase in mean reward. However, it struggled for almost 15,000 episodes to achieve a performance comparable to A3C, with a mean reward amounting to roughly -5. Eventually, the mean reward of the A3C reached a steady value of - 2.9 after 12,000. Meanwhile, the PPO continued aggressively exploring around 20,000 episodes to reach a performance similar to that of A3C. Between 20,000 and 25,000 steps, A3C and PPO algorithms sought rather conservative changes in suggested solutions and achieved slightly better mean rewards.

By 25000 episodes, both algorithms were exposed to the second scheduling problem. On average, the algorithm suffered considerable loss in mean reward over multiple experiments, but not all of them. It is depicted as distortions in the figure. The lines depict the mean rewards over multiple DRL experiments. Here, it was of particular importance to investigate the loss of performance and duration of recovery by every algorithm. Based on the analysis, the algorithms successfully recovered and achieved stable mean rewards after roughly 35,000 episodes for solving the second problem instance. Finally, we exposed the DRL agents to the most complex scheduling problem at the 50,000 episodes. Clearly, both algorithms suffered significantly in performance, which can be observed by the steep dive of blue and orange lines in the figure. On average, both algorithms successfully started recovering and reporting higher mean rewards during the experiments. However, A3C once again showed that it is able to recover and deliver high-quality solutions for solving problems compared to PPO.

Similarly, we conducted ten DRL experiments using each algorithm to address the same problem instances and investigate the second encoding. We have given DRL algorithms full control over the heuristic library component. They selected allocation and sequencing algorithms at every decision point during a scheduling period. The obtained results of the experiments are depicted in Figure 4.21.

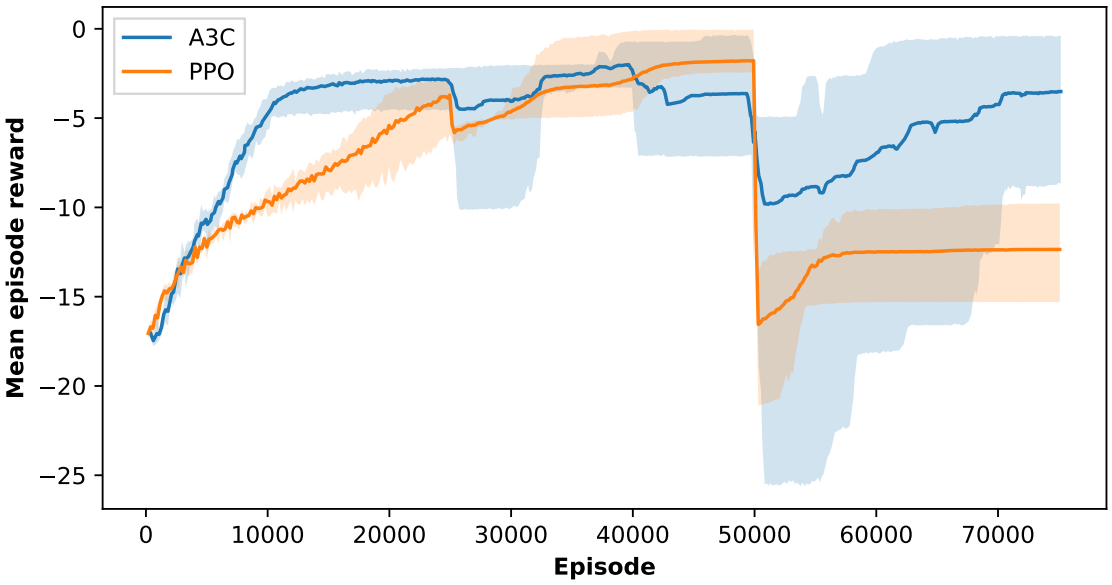


Figure 4.21: Computational results on the performance of MESEAS-A3C and MESEAS-PPO using the allocation and sequencing approach (Nahhas et al., 2022b).

The allocation and sequencing formulation of the DRL scheduling model is significantly more complex than the previous one. In simple terms, the number of possible actions in the actions space was doubled. Following identical configuration, both algorithms started searching for high-quality solutions for the first problem and kept doing so for 25,000 episodes. Then, we provided the algorithm with the second problem, and finally, by episode 50,000, they accessed and started solving the third problem instance. As shown in the figure, the achieved mean reward by the A3C started steadily increasing over training episodes. However, it was evident that the algorithm's convergence took considerably longer and started by episode 10,000. As for the second algorithm, the PPO started similarly maximizing its mean reward for solving the first problem. However, the results showed that further training is required to achieve a stable and high mean reward compared to A3C. The behavior of the PPO using the second encoding was somewhat similar to the first one. After changing the scheduling problems, we recorded worse mean rewards by both algorithms after each change and a steady recovery afterward. Based on our formulation of the DRL scheduling problem, A3C again outperformed PPO in terms of maximizing the mean reward and, thereby, the objective values for solving the scheduling problem. It also proved, on average, quicker training and more robust policy recovery after changing the considered scheduling problems.

In summary, the computational results of both experiments demonstrated that the developed DRL scheduling models and the adopted DRL methods are able to learn from solutions to scheduling problems. Despite performance suffering when a considered problem was changed significantly, the presented solutions are able to recover and achieve high mean reward. It implies that both algorithms, despite the demonstrated overall better performance of A3C, can adjust their learned policies and adapt to significant changes in the scheduling problem considered. Increasing the complexity of the actions space by

allowing DRL agents to control another set of heuristics required the algorithm to be trained longer to achieve comparable performance to previous experiments. Usually, DRL methods are trained to deal with very simple tasks formulated using a fairly simple action space, unlike our presented setup. The loss in performance, though, hinted at insufficient generalization in the behavior of the algorithms. Our observations largely agree with the findings presented by Zhang et al. (2018). It is a well-recognized issue in DRL and requires further investigation of them from fundamental and application perspectives.

4.5.5 Summary and comparison to related work (Eval 3.3 - partial Eval 4)

In this section, we started with a brief overview of the considered multi-stage scheduling environment that manufactures PCBs with an HFS structure. Afterward, we instantiated the presented methodology $\text{MESEAS}_{\text{multi}}$ to find solutions for a set of real problem instances PI_{multi} that minimizes a set of objective business values Γ. Throughout the instantiation, we utilized the presented scheduling model SP_{multi}, the simulation component, the optimization component, and the machine learning component. We conducted extensive experimental evaluations to verify the full integration of these components and their provided functionalities. The presented methodology achieved significant improvement in solving the scheduling problems and suggested high-quality solutions that minimized the considered objective values compared to conventional techniques.

Our investigation of conventional machine learning techniques showed that they are suitable for addressing descriptive and predictive business problems (Kharitonov et al., 2022). As for prescriptive business needs, such as scheduling problems, we adopted deep reinforcement learning techniques to design and develop the machine learning component of the artifact. To establish a comparison to related works, we instantiated MESEAS for solving the two-stage scheduling problem discussed in Lang et al. (2020). The authors investigated two stages with five identical machines in the first stage and four identical machines in the second stage $\langle HFS_2\ (P_5,\ P_4) \mid f_{g,h},\ D_{pr} \mid\ T\rangle$. The problems are solved to minimize the total tardiness. In Lang et al. (2020), a novel adoption of Artificial Neural Network-based (ANN) machine learning for solving scheduling problems was presented. The authors adopted NeuroEvolution of Augmenting Topologies (NEAT) originally proposed by Stanley and Miikkulainen (2002) and further described in Miikkulainen and Stanley (2009). NEAT is a hybrid technique that combines the use of GA to optimize the topologies and parameters of ANN to solve a given problem.

In Lang et al. (2020), NEAT was used to approximate the sequence of jobs to be scheduled on a set of machines to minimize total tardiness. The computational results for solving four problem instances suggested that the presented technique after training can be used for job scheduling to minimize total tardiness. However, certain limitations in terms of minimizing other objective values were open for further research. Two improved variations of the presented NEAT and an extended evaluation were presented in (Lang et al., 2021b). The improved variations of NEAT yielded better total tardiness in the first and second problem instances at the expense of a relatively worse makespan.

Table 4.6 presents the collected computational results of MESEAS-Multi and MESEAS-Single for solving four problem instances from the considered scheduling environment. The results are compared to NEAT in terms of minimizing the makespan, the total major setup times, the total tardiness, and total penalties (Lang et al., 2020). The authors did not report on the number of penalties, and therefore, we left these values unfilled. Both proposed techniques in MESEAS outperformed NEAT in minimizing the total major setup times and total tardiness. MESEA-Single, though, optimized the total major setup times at the expense of the makespan objective compared to NEAT and somewhat MESEAS-Multi. MESEAS-Multi dominated NEAT for minimizing all considered objective values, as highlighted in the table in light gray. MESEASE-single delivered better results in terms of minimizing the total major setup times.

Table 4.6: Evaluation results of optimization compared to NEAT presented in (Lang et al., 2021b) for solving two-stage scheduling problems (Nahhas et al., 2022a).

	MESEAS-Multi				**MESEAS-Single**				**NEAT**			
PI	$\mathbf{C_{max}}$	**MS**	**T**	**U**	$\mathbf{C_{max}}$	**MS**	**T**	**U**	$\mathbf{C_{max}}$	**MS**	**T**	**U**
1	16,888	50	0	0	17,786	43	0	0	17,768	114	124	–
2	19,942	51	0	0	22,458	53	0	0	20,916	149	303	–
3	18,405	58	0	0	22,671	53	0	0	20,584	142	0	–
4	18,393	48	0	0	22,569	45	0	0	18,771	113	0	–

Table 4.7 presents the evaluation of the DRL agents MESEAS-A3C and MESEAS-PPO for solving the two-stage problem instances. The results are also compared to NEAT in terms of considered objective values. The best obtained objective values are highlighted in the table with light gray filling. Both proposed DRL methods in MESEAS outperformed NEAT in minimizing the total major setup times and the makespan. The results of MESEA-A3C fully dominated NEAT for solving the first three problem instances. MESEAS-PPO outperformed MESEAS-A3C in terms of minimizing the total major setup times. However, both methods were dominated by NEAT for solving the fourth problem instance in terms of minimizing the makespan, the total tardiness, and the total number of penalties. On average, MESEAS-A3C delivered the highest quality solutions for solving the problem instances with a slight deviation in terms of total major setup times compared to MESEAS-PPO.

We again observed that the performance of MESEAS-A3C and MESEAS-PPO may decrease in minimizing some objective values. That is evident in their performance in solving the fourth problem instance. According to the author's analysis, the fourth problem is, in fact, the least complex problem, with only 143 jobs to be scheduled (Lang et al., 2020, p. 1301). The first three problem instances had, on average, 170 jobs to be scheduled. This change in the nature of the problem could explain the behavior of the algorithms.

Table 4.7: Evaluation results of machine learning compared to NEAT presented in (Lang et al., 2021b) for solving two-stage scheduling problems (Nahhas et al., 2022b).

	MESEAS-A3C				**MESEAS-PPO**				**NEAT**			
PI	$\mathbf{C_{max}}$	**MS**	**T**	**U**	$\mathbf{C_{max}}$	**MS**	**T**	**U**	$\mathbf{C_{max}}$	**MS**	**T**	**U**
1	17,081	58	0	0	18,609	54	0	0	17,768	114	124	–
2	17,669	55	0	0	17,874	55	0	0	20,916	149	303	–
3	17,690	56	0	0	18,228	54	0	0	20,584	142	0	–
4	19,767	52	1,627	4	19,970	52	2,245	4	18,771	113	0	–

4.6 Summary of MESEAS evaluation

This chapter started with an introduction to the adopted evaluation method (Sonnenberg and vom Brocke, 2012). The first section summarized the first evaluation activity based on Sonnenberg and vom Brocke (2012), in which the problem statement and research objective of the research were outlined. Subsequently, the second evaluation activity was recapped in the second section. This section highlighted the consistency and systematic approach in selecting the design tools that are used to present the design blueprints of the main components of MESEAS. The third evaluation stage comprised three evaluation activities that are detailed in the third, fourth, and fifth sections of this chapter.

First, we evaluate the presented methodology by means of proof of concept (von Hevner et al., 2004). The third section was dedicated to discussing the implementation and deployment of the proof of concept. We concluded in this section that the adoption of open-source and cloud-native technologies was decisive in meeting the functional and non-function requirements of the artifact. Their adoption and utilization facilitated a scalable, hybrid, and multi-architecture deployment of the artifact. Such deployment ensured the efficient operation of MESEAS, taking into account energy consumption. The simulation and heuristic subsystem, as well as the optimization and machine learning subsystem, were developed and deployed with parallelization and scalability features. This section concluded with a summary of the presented proof of concept and its operation, emphasizing effectiveness and efficiency.

Then, we conducted evaluation experiments on multiple use cases to address single-stage scheduling problems in cloud environments. The evaluation results were presented in the fourth section, which detailed the instantiation of the presented methodology to address scheduling concerns in a cloud environment. Finally, the section summarized the findings of the MESEAS evaluation using three benchmarks from the literature. Subsequently, the fifth section discussed the performance evaluation of the presented methodology for solving real multi-stage scheduling problems in manufacturing. The section was summarized with comparisons to related works focusing on the performance of MESEAS optimization and machine learning components. On average, the scheduling solutions of the optimization component were 32.6 % superior to those of the related work in terms

of minimizing system efficiency-related objective measures. The achieved solutions by the machine learning component similarly dominated the solutions of the related works with a margin amounting to 31.9 %. Given the improvement in solving real problem instances, the evaluation that was conducted verified the usefulness of MESEAS for addressing real multi-objective scheduling problems.

5 Conclusion and future research perspectives

5.1 Summary and discussion

In the context of this work, a scheduling methodology was developed to integrate the utilization of simulation, heuristic, improvement, and deep reinforcement learning methods to address multi-objective scheduling problems. The objective of their integration is to leverage their combined potential and mitigate their limitations. Following modular design practices, MESEAS architecture incorporates simulation, heuristic library, optimization, and machine learning components that are integrated to address single-stage and multi-stage scheduling problems. The capabilities of MESEAS components can be accessed and utilized through four distinct functionality layers.

The modeling and simulation layer allows decision-makers to investigate what-if scenarios and execute various analyses to make informed capacity planning decisions in the context of scheduling. This layer integrates the presented scheduling model with an automated simulation model composer to build and execute simulation models automatically. The artifact's simulation component is based on the discrete event simulation paradigm used to construct simulation models, which are executed using open-source simulation engines. The presented scheduling model and the simulation component of MESEAS fulfill the *first*, *second*, and *third* functional requirements of the artifact.

The constructive heuristic layer offers decision-makers further functionalities to explore basic solutions for a given scheduling problem, taking into account some business objectives. This layer is fully integrated with the underlying modeling and simulation layer and produces solutions for scheduling problems after selecting some allocation and/or sequencing algorithms. In this work, we developed and adopted many allocation and sequencing heuristics and many other well-established heuristics from the literature. The developed and adopted allocation and sequencing heuristics in the heuristic library components alongside the scheduling model and the simulation component of MESEAS fulfill the *fourth* and *fifth* functional requirements of the artifact.

The optimization layer facilitates the systematic examination of the solution space of a scheduling problem. It offers decision-makers improved, high-quality solutions that consider multiple business objectives, such as system efficiency and customer satisfaction. The optimization layer integrates and harmonizes the combined utilization of simulation, heuristic, and improvement methods. This integration was achieved by developing several encoding models, which grant an improvement method full control over a set of selected

allocation and sequencing heuristics to explore improved solutions for scheduling problems. The adopted evolutionary improvement methods alongside the presented encoding models of the optimization component and their integration with underlying components of MESEAS fulfill and improve the *fourth*, *fifth*, and *sixth* functional requirements of the artifact.

Finally, the machine learning layer extends the available functionalities and grants decision-makers access to deep reinforcement learning methods to investigate improved and adaptive solutions for multi-objective scheduling problems. The machine learning layer integrates the functionalities of all underlying layers and their components. It is integrated through the presented DRL scheduling and evaluation models. The developed DRL scheduling and evaluation models permit the adopted DRL algorithms to control allocation and sequencing heuristics during a scheduling period to solve multi-objective scheduling problems. The adopted DRL methods based on the developed DRL scheduling and evaluation models of the machine learning component fulfill the *sixth* and finally the *seventh* functional requirements of the artifact.

After designing and integrating the individual components in MESEAS, we implemented the final artifact using cloud-native technologies. We relied on open-source tools and frameworks to ensure efficient deployment and operation of MESEAS subsystems and their underlying components. The artifact can be deployed and hosted on multi-architecture hardware, including ARM-64 and x86-64 physical devices. Multi-architecture deployment facilitates the flexible operation of MESEAS and allows an engineer to either rely on power-conservative hardware for efficiency or powerful computing hardware for instant decision-making. In addition to the layered architecture, this implementation substantially contributes to the flexible and adaptive functionality of the artifact, which fulfills the *third*, and partially *fourth* indicated non-functional requirements.

The component-based design and the adoption of cloud-native technologies were the cornerstones in the design and implementation of the artifact to achieve scalable, distributed, flexible, and adaptive functionality of the artifact (cf. Chapter 3 and Section 4.3). The workloads of simulation, optimization, and machine learning components are parallelized and fully distributed to achieve high-quality solutions for scheduling concerns in a reasonable time as demonstrated in Section 4.3. The artifact can be deployed on in-house hardware and further scaled up or down using cloud infrastructure. The scalability feature offers decision-makers the prospect of on-demand computing resources to operate the artifact. The modular design of the presented methodology substantially simplifies adapting and incorporating new heuristic, improvement, or DRL methods into the artifact. Despite implementing the most significant operational constraints and business objective measures in the presented scheduling model, it can be extended and customized with minimal effort to address changes in business needs. In summary, the presented artifact fulfills the *first*, *second*, and fully the *fourth* indicated non-functional requirements.

The overreaching objective of this work is to present a novel scheduling methodology that harmonizes the integrated utilization of simulation, heuristic, improvement, and deep reinforcement learning methods to deal with multi-objective scheduling problems. This

objective was based on three formulated hypotheses, presuming the conceptualization, the design, and the potential of their integrated use for solving complex multi-objective scheduling problems. We also argued that deep reinforcement learning methods can be trained to learn from solutions to scheduling problems. To achieve this objective, following the DSR framework, we formulated an overarching research question: *"How to conceptualize, design, and develop a scheduling methodology that integrates and facilitates the combined utilization of simulation, heuristics, improvement, and deep reinforcement learning methods to address multi-objective scheduling problems in cloud and manufacturing environments?"*. To systematically investigate this research question and accomplish the overall objective, we decomposed the main research question and formulated four sub-questions. Answering each of them yields answering the overall question, starting with the first question:

First sub-research question - *What are the key methods frequently adopted to address multi-objective scheduling problems, and how do they perform in cloud and manufacturing environments?*. In this thesis, we conducted a literature analysis of conventional methods that are popular for solving scheduling problems. The results suggested that heuristic and improvement methods are dominant due to the complicated nature of the scheduling problem. The majority of them are NP-Hard optimization problems. Based on the literature analysis, we investigated and evaluated the performance of some popular heuristics and improvement methods in addressing single-stage and multi-stage problems.

After concluding the analysis of conventional methods, we conducted a structured literature analysis to investigate the adoption of the DRL method and its combination with other methods. The overall results of the investigations are thoroughly discussed throughout the second and third chapters of this thesis (cf. Chapter 2 and Chapter 3). In summary, the analysis demonstrated that conventional methods are either computationally expensive or fall short in terms of solution quality in addressing complex multi-objective scheduling problems. Advanced methods such as DRL are overwhelmingly adopted and evaluated in game-like environments. The structured literature review on their adoption to address scheduling problems yielded contributions dealing with simple scheduling problems. The identified related works were not integrated with heuristic and improvement methods as presented in Subsection 2.5.4 and Subsection 2.5.5.

Second sub-research question - *How do we efficiently combine heuristic and metaheuristic methods for solving scheduling problems in cloud and manufacturing environments?*. Integrating methodologically different methods, such as heuristic and improvement methods, requires the development of flexible encoding and decoding optimization models for scheduling. In the presented methodology, the optimization encoding models are foundational constituents for efficiently integrating and utilizing these methods. We presented and discussed three scheduling encodings. The computational results demonstrated that multi-population encoding, in which two populations coevolve to solve scheduling problems, delivers superior results in terms of minimizing the objective values. The encoding

scheduling models operate as interfaces between the components and facilitate accurate and efficient logical integration. That, in turn, facilitates their collaborative utilization for solving complex problems.

In addition, the simulation component of MESEAS is critical for improvement methods. It translates suggested solution individuals of improvement methods into actual scheduling solutions for a given problem. In essence, the simulation component is crucial for evaluation and solution construction. Overall, the component-based design and implementation are prerequisites for proper execution and efficient utilization. We discussed in depth the design and development of the simulation, heuristic library, and optimization components in Section 3.4, Section 3.5, and Subsection 3.6.2. The summaries of design evaluation were discussed in Subsection 3.4.5, Subsection 3.5.3, and Subsection 3.6.5. We consequently discussed the overall evaluation of the artifact in Chapter 4.

Third sub-research question - *Will a scheduling methodology that facilitates the combined utilization of heuristic and improvement methods outperform their individual use for solving multi-objective scheduling problems in cloud and manufacturing environments?*. The conducted evaluation of the presented methodology for addressing scheduling problems in cloud and manufacturing environments demonstrated significant potential in utilizing different heuristics over a scheduling period. The presented methodology adapts to changes in workload patterns and leverages the suitability of different heuristics to schedule jobs under different circumstances.

Utilizing several heuristics controlled by an improvement method in cloud environments delivered solutions with improved objective values in three independent use cases with different workload traces. The potential improvement in addressing single-stage scheduling problems increased in correlation to an increase in problem complexity. In the first use case in cloud environments, the instantiation of the methodology demonstrated an overall improvement in minimizing objective values compared to heuristic methods, amounting to over 50 %. In the second use case, a similar tendency was observed, starting with improved operation by 30 % and reaching up to 47 % better objective values compared to several heuristic methods. We discussed the results of the conducted evaluation extensively in Section 4.4.

Similar improvement was achieved in addressing more complex multi-stage scheduling problems in a manufacturing environment. The computational results indicated that the multi-population of MESEAS achieved significant improvement in terms of minimizing all objective values compared to heuristic and improvement methods. For instance, the EDD heuristic method, combined with simulation, achieved the best results in minimizing the considered objective values compared to other heuristic methods such as SPT or LPT. Since SPT and LPT lack any consideration in their rationale, the minimization of total tardiness or penalties, for that matter, the computational results of their performance, is not discussed.

Hence, compared to a heuristic method using simulation-based EDD, MESEAS-multi recorded, on average, a 116.4 % improvement in minimizing system efficiency objective

values in solving thirty real problem instances. System efficiency business objectives include the minimization of the makespan and total major setup times. Regarding customer satisfaction objective values, MESEAS-multi fully dominated the simulation-based EDD and fully minimized the total tardiness and the number of penalties in solving all problem instances. The simulation-based EDD heuristic method failed in 26.7 % of solved problem instances to eliminate total tardiness and recorded violations in eight problem instances. Compared to MESEAS-GA, using the attribute marker encoding model, MESEAS-Multi found scheduling solutions that are, on average, 11.7 % superior in terms of minimizing the makespan and the total major setup times. An instantiation of the methodology to solve problems from related works confirmed the findings. MESEAS-multi achieved superior performance compared to related works for solving other problem instances with an average margin that amounted to 41.3 %. The statistical analysis and computational results of the conducted experiments for solving multi-stage problems are profoundly discussed in Section 4.5.

Fourth sub-research question - *How to adopt deep reinforcement learning methods to learn from solutions to scheduling problems in cloud and manufacturing environments?*. To adopt DRL methods for addressing scheduling problems, scheduling environments are exposed to agents in a game-like manner. We relied on our collected findings and observations to integrate heuristic and improvement methods and followed a similar strategy.

We developed and presented flexible DRL scheduling and evaluation models. These models play a significant role in efficiently integrating DRL methods with simulation, heuristic, and improvement methods. They coordinate logical communication between DRL methods and the other integrated methods in MESEAS to solve scheduling problems. Based on them, DRL methods fully control available selected allocation and sequencing heuristics to solve scheduling problems. The simulation component was used to translate actions taken by DRL agents to construct actual scheduling solutions that minimize the considered objective values. Propagating back the constructed scheduling solution, in addition to key performance indicators, demonstrated that DRL can be trained to learn from solutions to scheduling problems. The DRL method's successful and efficient adoption was attributed to the artifact's flexible and modular design. Furthermore, relying on open-source frameworks eased the adoption of parallelization techniques for training. The rationales of the DRL scheduling and evaluation models were thoroughly discussed in Subsection 3.6.3.

To establish a baseline comparison to related works, we instantiated the methodology using DRL methods to solve problem instances of a multi-stage scheduling environment in the literature. The numerical results demonstrated that the proposed adoption of DRL methods using several DRL algorithms achieved improved solutions in minimizing the objective values. On average, the attained solutions using the MESEAS-A3C DRL improved objective values by 32,6 %. With a slight difference, MESEAS-PPO demonstrated a 30.8 % improvement compared to the published best results in the literature. A detailed analysis of the conducted evaluation into the performance of the machine learning component

of MESEAS was presented in Section 4.5. A condensed summary of the evaluation and the comparisons to related works were discussed in Subsection 4.5.5.

Contribution to answer the overarching research question
The scheduling methodology was conceptualized following component-based practices. It was designed using UML information flow diagrams, UML component and sequence diagrams, mathematical models and notations, and pseudocode to describe the rationale of developed algorithms. It was developed using cloud-native technologies and open-source frameworks to ensure efficient and accurate deployment and operation of the artifact. The combined utilization of simulation, heuristic, improvement, and deep reinforcement learning methods was achieved by developing a scheduling data model, several optimization encoding models for scheduling problems, DRL scheduling models, and a DRL evaluation model. The developed scheduling data model facilitates flexible instantiation of the methodology to address single-stage or multi-stage scheduling problems considering multiple objective measures. The problems can be solved using either a pure or a weighted-sum multi-objective formulation.

Answering the research questions verified the articulated *first*, *second*, and *third* investigated hypotheses of this thesis. Verifying expressed presumptions and answering the overarching research question yields meeting the overall objective of this research. In the previous chapter, we highlighted the limited evaluation in the cloud environment, which would meet the requirements of the third evaluation stage based on (Sonnenberg and vom Brocke, 2012). In manufacturing, we conducted several evaluation activities in the fourth stage to partially meet the concept of three realities conditioned by the adopted evaluation method (Sonnenberg and vom Brocke, 2012). Full integration of the proposed artifact in an organizational context is too time-consuming and, unfortunately, was not fully achieved. Despite these shortcomings and other limitations, which we discussed in previous chapters, this thesis presented several contributions; these are:

- A comprehensive review and analysis of conventional solutions to address scheduling problems in cloud and manufacturing environments.
- A comprehensive review of modern solution methods to address scheduling problems in cloud and manufacturing environments.
- The design and development of several allocations and sequencing heuristic methods to address scheduling problems in cloud and manufacturing environments.
- The design and development of overarching mathematical and data models for scheduling problems that include the most significant operational constraints and objective measures. These models are flexible and extendable to address further scheduling concerns.
- The design and development of several optimization encoding models for scheduling problems, which include conventional (MESEAS − GA), hybrid (MESEAS − single), and multi-population model (MESEAS − multi).

- The adoption, integration, and evaluation of several single-objective and multi-objective metaheuristics for solving scheduling problems (GA - NSGA III).
- The design and development of DRL scheduling and evaluation models, which encode scheduling problems and decode scheduling solutions for adopting DRL methods to address scheduling concerns.
- The adoption of the MDP mathematical framework to map and integrate the presented scheduling model, simulation component, and the heuristic library components of the artifact with DRL methods.
- The adoption, integration, and evaluation of several prominent DRL methods for solving multi-objective scheduling problems (A3C, PPO, MARWIL).
- The conceptualizing, design, development, and evaluation of an overarching scheduling methodology that harmonizes and facilitates the combined utilization of simulation, heuristic, metaheuristic, and DRL methods for solving multi-objective scheduling problems.
- The development and deployment of the component-based design of the artifact, which encompasses modeling, simulation, heuristic library, optimization, and machine learning components. Adopting cloud-native technologies and other open-source frameworks facilitates an efficient multi-architecture deployment and execution of the artifact.

The artifact presented in this thesis was evaluated to address several variations of scheduling problems, mainly in cloud and manufacturing environments. For instance, we conducted initial experiments to address scheduling problems in the supply chain (Nahhas et al., 2023b) and recently presented further insights in (Nahhas et al., 2024b). In this thesis, we focused on two use cases in cloud environments and two use cases in manufacturing environments. Table 5.1 and Table 5.2 summarizes the main findings of the discussed use cases. In Table 5.1, we highlighted the nature of the considered problem, the used baseline for comparison, and the utilized method in the presented methodology. Table 5.1 highlights the average achieved improvement using MESEAS compared to baselines in terms of minimizing the objective values. The table also illustrates the evaluation stage, which was covered in every use case. We rely on the evaluation stage metric to discuss the methodology's maturity for real-world use in the considered use cases.

To address scheduling concerns in the first use case in a cloud environment, we developed two scheduling heuristic methods with several rescheduling mechanisms and integrated them with a metaheuristic method. The EA-FFD and the WA-FFD heuristic methods were used as baselines and compared to MESEAS-single. In the second use case in a cloud environment, we adopted and integrated several heuristic methods from related works. We compared them against MESEAS-single to address scheduling problems using a known benchmark. The summary of these two use cases is presented in the first three rows in Table 5.1.

To address scheduling problems in the first use case in manufacturing, we developed several allocation and sequencing heuristic methods and combined them with metaheuristic and DRL methods. We compared the performance of the presented methodology in minimizing multiple objective values against the simulation-based EDD method and the GA method. In essence, we utilized heuristic and metaheuristic methods as baselines for the comparison. To establish a comparison to related works, we instantiated the presented methodology using MESEAS-Multi, MESEAS-A3C, and MESEAS-PPO to address multi-objective scheduling problems in a second use case. In this use-case, we used the reported results on the performance of NEAT presented by Lang et al. (2021b) as a baseline for the comparison. The summary of these two use cases in manufacturing is presented in the second part of Table 5.1.

The achieved improvements in minimizing the considered objectives compared to the discussed baselines are presented in Table 5.2. In cloud environments, we considered mainly the minimization of energy consumption (E) and the total penalties or violations in SLAs (U). In the first use in cloud environments, the overall computational results demonstrated that MESEAS-single fully dominates the EA-FFD in minimizing the considered objective values. It outperformed the heuristic by 3.46 % and 42.91 % in minimizing the energy consumption and total penalties, respectively. Compared to WA-FFD, MESEAS-single lost in performance by -2.69 % in terms of minimizing energy consumption but reduced the total number of penalties by 54.1 %. The detailed percentage calculations and the comparison with the heuristic methods are presented in Section B.1

In the second use in cloud environments, the collected results showed that, on average, MESEAS-single minimized the overall energy consumption by 53 % while losing performance by -1.0 % in terms of penalties compared to the baseline heuristic methods in the literature. We computed these percentages by averaging the achieved improvement compared to the baseline methods. The detailed achieved improvements compared to every heuristic are presented in Subsection B.2.2.

The achieved improvements in manufacturing use cases are presented in the lower part of Table 5.2. In manufacturing environments, we considered the minimization of the makespan (C_{max}), the number of major setup times (MS), the total tardiness (T), and the total number of penalties (U). MESEAS-multi fully dominated heuristic and metaheuristic methods in minimizing all objective values in the first use case. We weighted system efficiency objective measures equally to compute the average improvement of the makespan and the total number of major setup times. We similarly utilized equal weights to compute the average of business objectives related to customer satisfaction. MESEAS-multi found scheduling solutions that minimized, on average, the makespan and the total number of major setup times by 40.5 % compared to simulation-based EDD. In terms of minimizing the total tardiness and number of penalties, MESEAS-multi found solutions were, on average, 26.7 % superior to those of simulation-based EDD. The improvement margin narrows down compared to the metaheuristic method. MESEAS-multi nevertheless fully dominated the GA in solving all problem instances by minimizing the considered objective values. It delivered, on average, superior solutions amounting to 10.1 % in

terms of minimizing the makespan and the major setup times. Regarding penalties and total tardiness, MESEAS-multi achieved better solutions by 3.3 % on average for solving all problem instances. Section C.2 comprises the detailed computations of the presented averages.

In the second use case in manufacturing, MESEA-multi fully dominated NEAT for solving all problem instances. It delivered scheduling solutions capable of minimizing the makespan of the major setup times, on average, by 32.6 %. Similarly, it dominated NEAT in solving 50 % of the considered problem instances in minimizing the total tardiness and number of penalties. MESEAS-A3C and MESEAS-PPO outperformed NEAT in minimizing the objective values, on average, by 31.9 % and 30.8 % in terms of minimizing the makespan and the total number of major setup times, respectively. On average, they achieved 25 % better solutions in terms of total tardiness and number of penalties.

Table 5.1: Evaluation summary of use cases (I).

Considered problem	Base-line	Utilized Method
P_m - Cloud	Heuristic method (EA-FFD)	MESEAS-Single
P_m - Cloud	Heuristic method (WA-FFD)	MESEAS-Single
Q_m - Cloud	Related works (Multiple)	MESEAS-Single
HFS_4 - Manufacturing	Heuristic method (EDD)	MESEAS-Multi
HFS_4 - Manufacturing	Metaheuristic (GA & DA-SI)	MESEAS-Multi
HFS_2 - Manufacturing	Related works (NEAT)	MESEAS-Multi
HFS_2 - Manufacturing	Related works (NEAT)	MESEAS-A3C
HFS_2 - Manufacturing	Related works (NEAT)	MESEAS-PPO

Table 5.2: Evaluation summary of use cases (II).

Method / objectives	E (%)	U (%)	Evaluation stage
MESEAS-Single	3.46	42.91	Eval 3
MESEAS-Single	-2.69	54.1	Eval 3
MESEAS-Single	53.1	-1.0	Eval 3
Method / objectives	C_{max} & MS (%)	T & U (%)	Evaluation stage
MESEAS-Multi	40.5	26.7	Eval 3 - Partial Eval 4
MESEAS-Multi	10.1	3.3	Eval 3 - Partial Eval 4
MESEAS-Multi	32.6	50.0	Eval 3 - Partial Eval 4
MESEAS-A3C	31.9	25.0	Eval 3 - Partial Eval 4
MESEAS-PPO	30.8	25.0	Eval 3 - Partial Eval 4

5.2 Outlook and further research directions

With the introduction and adoption of new technologies, scheduling problems in cloud and manufacturing environments become more complex. In this thesis, we presented a scheduling methodology to solve multi-objective scheduling problems. We evaluated its utility in addressing scheduling problems in manufacturing and cloud environments over multiple use cases. Although the presented methodology outperformed heuristic, metaheuristic, and machine learning-based methods, the individual methods of the artifact are subject to certain limitations from a research and engineering perspective.

From an engineering perspective, security and data management are further aspects that must be addressed. For instance, the artifact was deployed with no holistic security concept in mind. Although relying on open-source technologies such as Kubernetes for the deployment elevates some of the security issues, further aspects, such as access management, must be addressed. In addition, despite the immense flexibility of document databases, in the MESEAS design, relying on a document database became inefficient for dealing with large amounts of data in a single collection. Thus, the efficiency of post-processing the results was slightly impacted. Hence, online analytics of the results in the front end could be improved. Here, a relational database would offer better performance for the user in conducting and executing complex online analyses of the best scheduling solutions. Finally, the current deployment of the artifact may be vulnerable to the failure of some critical components, such as messaging systems or other cluster and networking components. Here, redundancy is required to secure a higher availability of artifact services.

From a research perspective, we will independently outline limitations and some research directions in the design of the optimization and machine learning components of the artifact. The presented methodology was designed and developed with flexibility in mind by splitting the core logical components and functionalities from each other. The motivation behind this foundational design decision was the diverse expertise and skills that are required to operate and maintain different types of solution methods. For many decision-makers, relying on heuristics that are simple to follow and generate reproducible scheduling solutions outweighs higher-quality solutions generated by advanced metaheuristics or deep reinforcement learning methods.

Metaheuristics or deep reinforcement learning methods require engineering and domain skills to be properly parametrized and tuned to achieve efficient and accurate scheduling solutions. Despite incorporating default configuration for tuning the optimization and machine learning components, significant changes in the scheduling environment may impact efficiency. That may lead to a higher computational effort to obtain high-quality scheduling solutions.

Therefore, automating hyperparameter tuning of the optimization component is a research direction that ensures robustness and increases the adaptivity of the presented methodology. We conducted a preliminary investigation on the topic and explored the potential of hyperparameter optimization methods in manufacturing environments in

(Chernigovskaya et al., 2024). The optimization component may also benefit from further investigations into the potential of the presented multi-population approach and its applicability to address further use cases.

As for the machine learning component, the evaluation of MESEAS-A3C and MESEAS-PPO demonstrated that agents may suffer performance loss if the scheduling problem significantly changes before recovery. These findings point to an insufficient generalization problem of the trained DRL agents. This generalization problem of DRL methods was also discussed and thoroughly investigated by Zhang et al. (2018). Several research directions are promising for mitigating this issue. For instance, we recently investigated the potential of imitation learning methods to achieve better performance. We trained a DRL agent based on the Monotonic Advantage Re-Weighted Imitation Learning (MARWIL) method (Wang et al., 2018). To train the agent with previous high-quality experiences, we utilized MESEAS-Multi to generate high-quality scheduling solutions. Using the same DRL scheduling and evaluation model, the computational results showed that the imitation learning approach achieved a higher mean reward amounting to 10 % compared to pure DRL.

Despite successfully training several DRL methods for solving multi-objective scheduling problems, the presented DRL scheduling model, especially the observation space, can be improved. The definition and the nature of the observation have a significant impact on the performance and the generalization of DRL methods. We recently investigated the potential of an image-based definition of the observation space to address scheduling problems in a retailing supply chain in (Nahhas et al., 2024a). The collected results demonstrated significant potential in addressing scheduling problems and may be further explored to achieve conclusive results.

A Graphical user interface

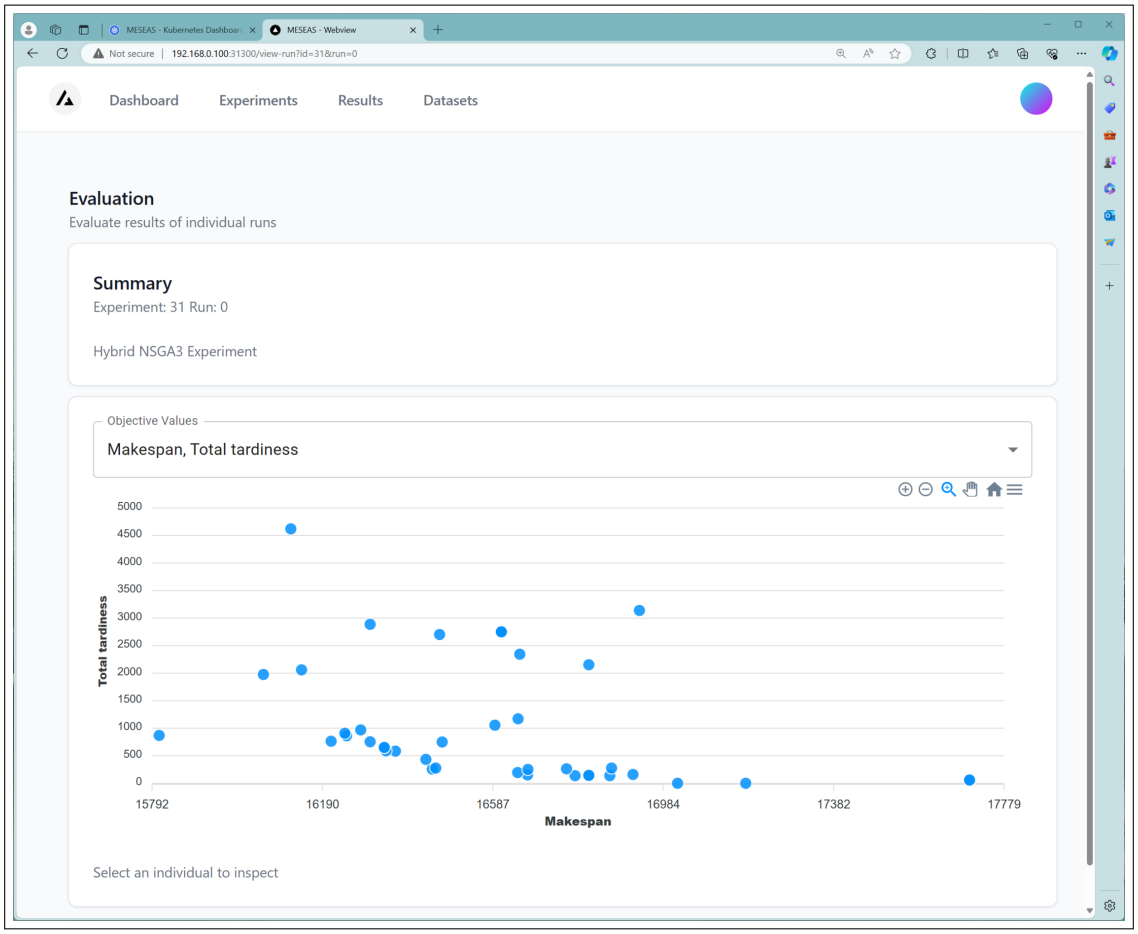


Figure A.1: User dashboard for results analysis: Best solutions and their makespan and total tardiness objective values.

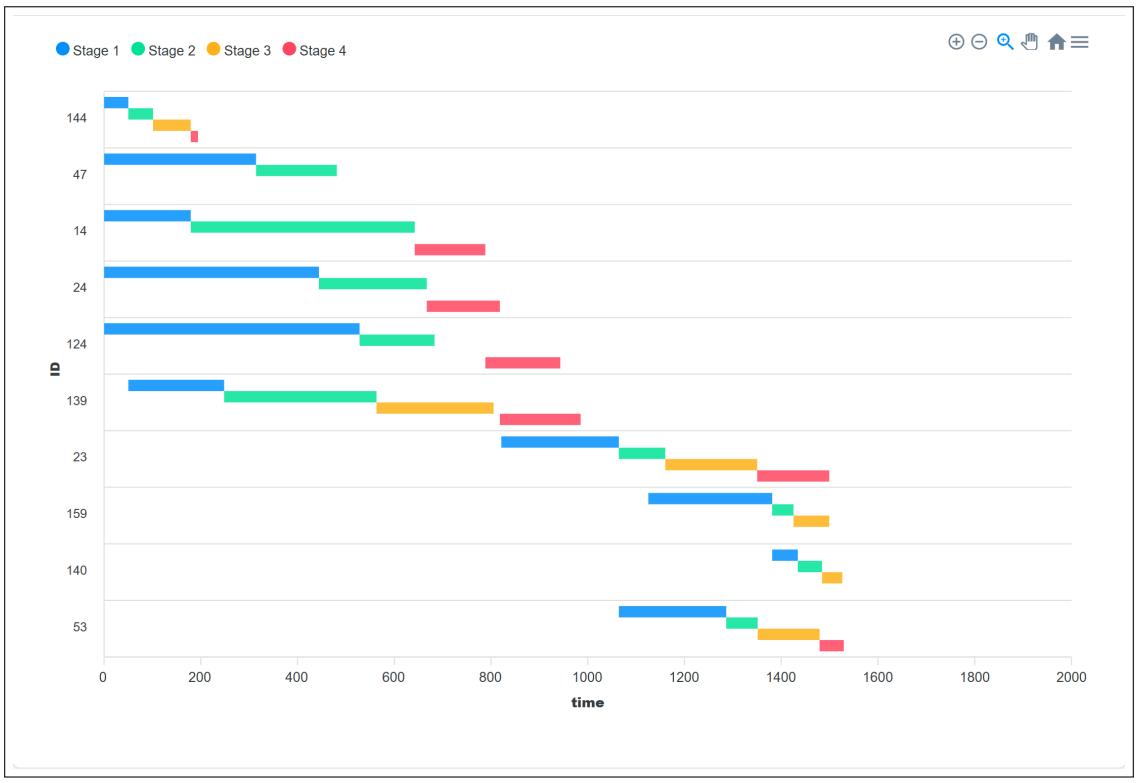


Figure A.2: User dashboard for results analysis: Gantt chart for bottlenecks analysis.

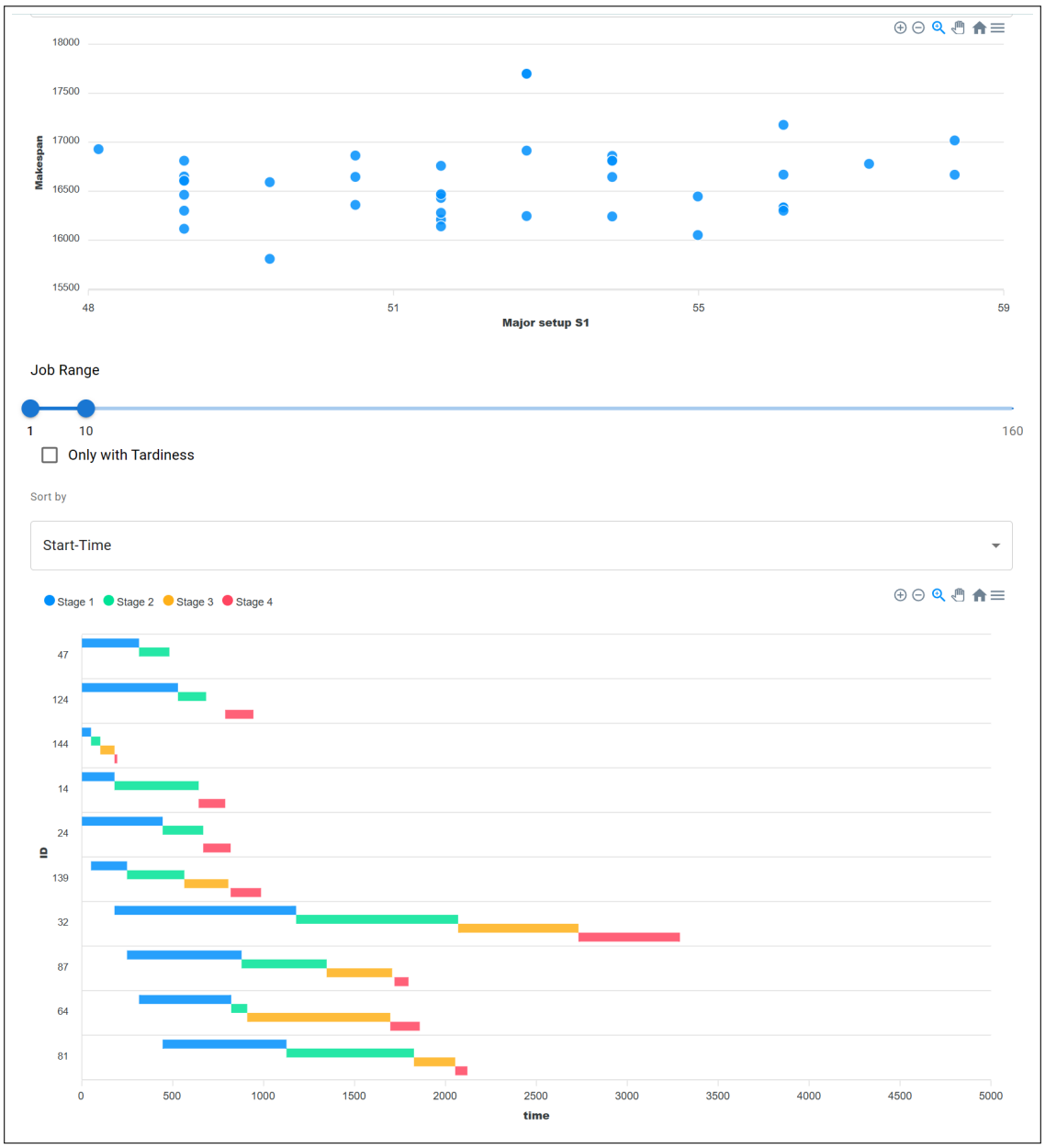


Figure A.3: User dashboard for results analysis: Best solutions and their makespan and major setup times objective values.

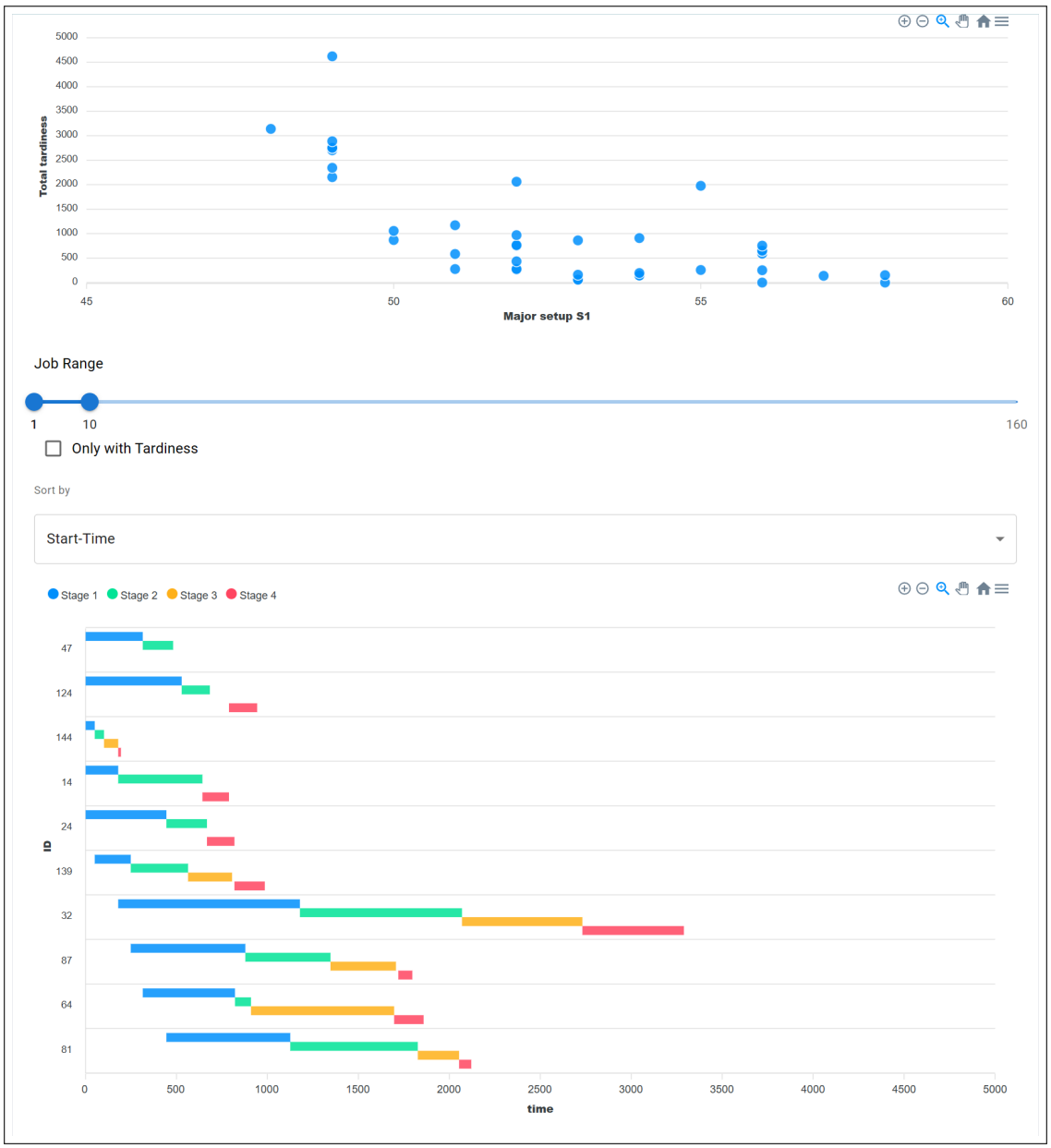


Figure A.4: User dashboard for results analysis: Best solutions and their total tardiness and major setup times objective values.

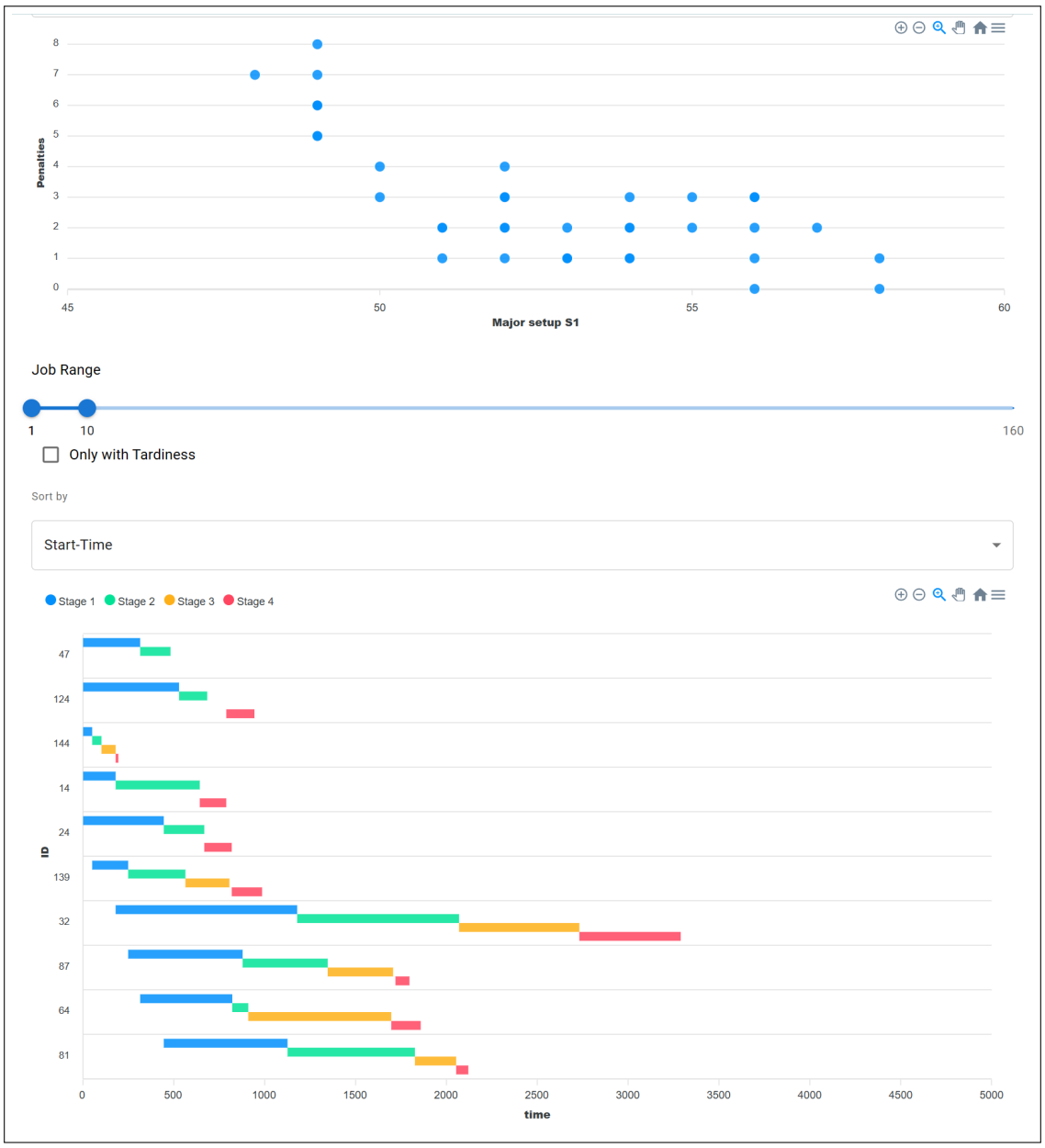


Figure A.5: User dashboard for results analysis: Best solutions and their major setup times and penalties objective values.

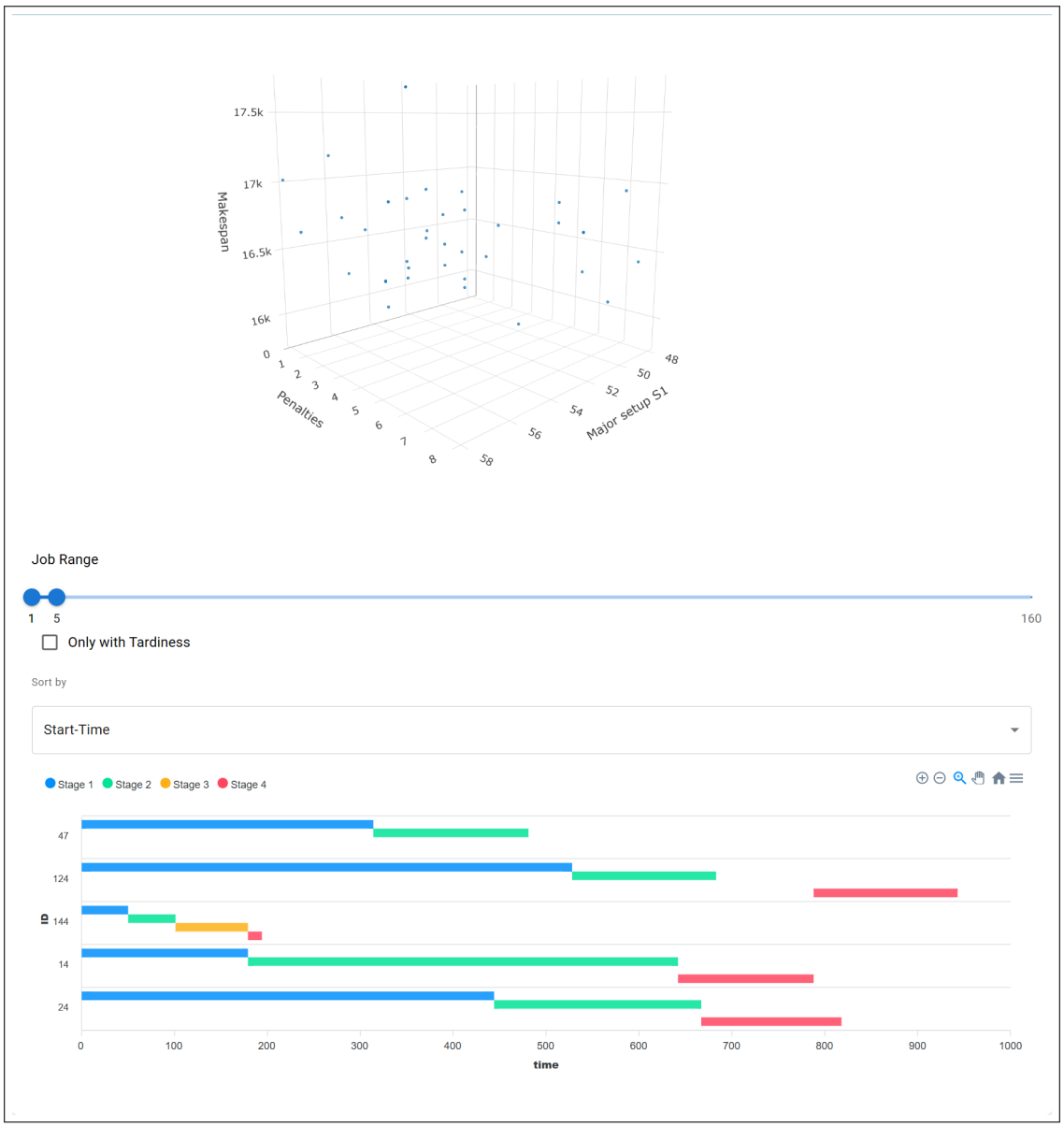


Figure A.6: User dashboard for results analysis: Example analysis of three objective values.

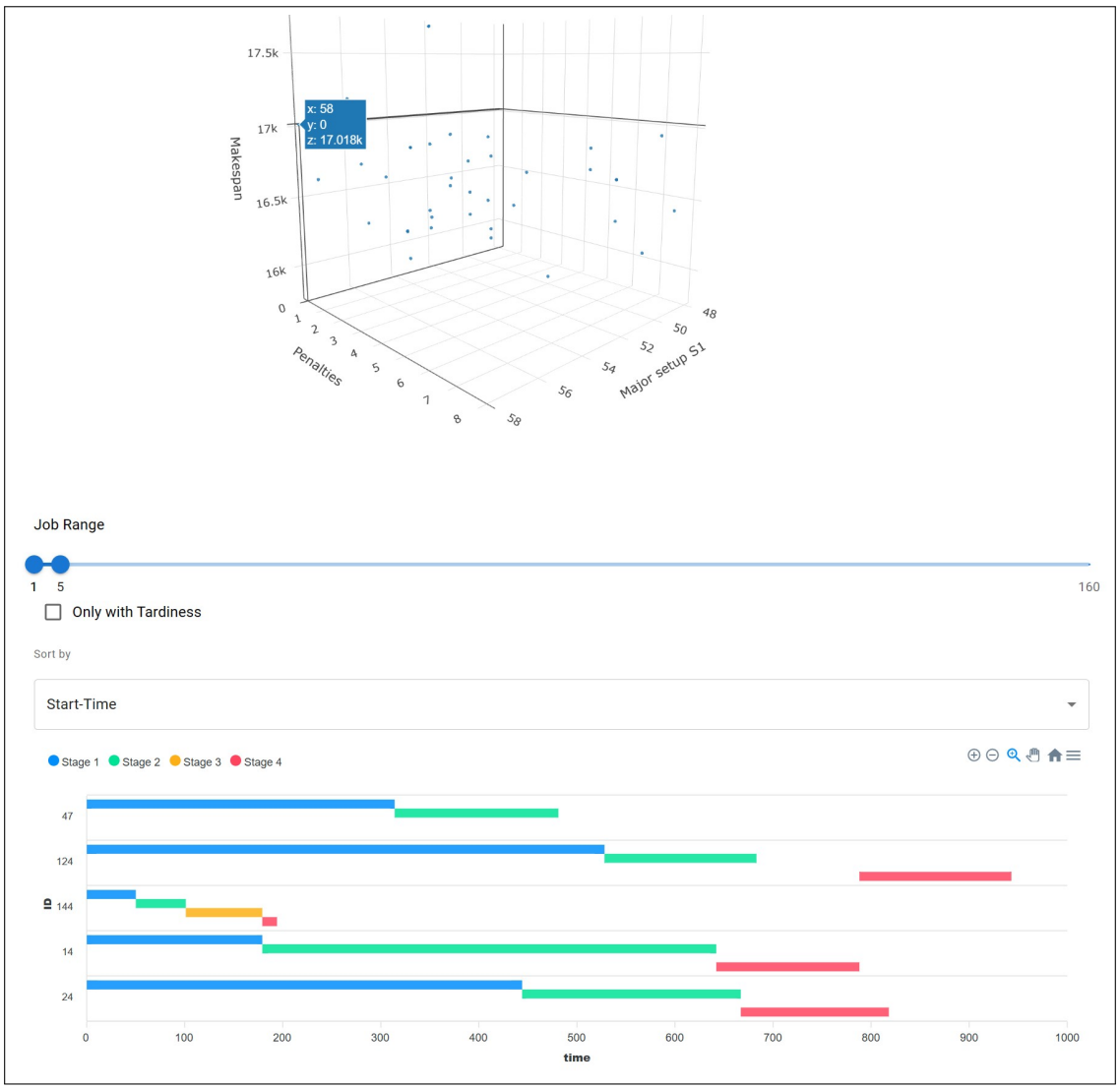


Figure A.7: User dashboard for results analysis: Example three objective value analysis view with the best solution.

B Scheduling in cloud environments

B.1 First use case: Performance comparison in percentages

Table B.1: Computational results of online hours and job rescheduling.

	Online hours[1]	Rescheduled jobs[2]	Initiated rescheduling
Is-situation	960	0	0
WA-FFD	253	342	22
EA-FFD	295	275	18
MESEAS-Single	272	157	18

Table B.2: Summary of computational results.

	Decrease in online hours (%)	Increase in rescheduled jobs (%)
Is-situation	0	0
WA-FFD	73.6	23.6
EA-FFD	69.3	19.0
MESEAS-Single	71.7	10.8

[1]The total available online hours overall physical machines during five days of operations.
[2]The number of jobs during the scheduling period is $n = 1450$.

Table B.3: Overall performance comparison between MESEAS-Single and heuristic methods in terms of minimizing the objective values.

Performance comparison between MESEAS-Single and Is-situation.

Objective value	Online hours (%)	Rescheduled jobs (%)
Average improvement	71.7	-10.8
Overall improvement	60.9 %	

Performance comparison between MESEAS-Single and WA-FFD.

Objective value	Online hours (%)	Rescheduled jobs (%)
Average improvements	-2.69	54.1
Overall improvement	51.41 %	

Performance comparison between MESEAS-Single and EA-FFD.

Objective value	Online hours (%)	Rescheduled jobs (%)
Average improvements	3.46	42.91
Overall improvement	46.37 %	

B.2 Second use case

B.2.1 Methodology instantiation and problem formulation

The presented methodology was instantiated to address scheduling concerns of the discussed use case as presented in Equation B.1. To address scheduling concerns in this environment, we utilized the first three functionality layers of MESEAS. The scheduling data model, the simulation, the heuristic library, and the optimization components are therefore instantiated. Given the scheduling problem SP_{single}, we utilize $MESEAS_{(X,\ T)}$ to find a solution $X \in \mathbb{X}$ that minimizes the objective function Γ_ω during the scheduling period T. In the course of this section, we will elaborate on the instantiation of the methodology.

$$MESEAS_{(X,\ T)} = \langle SP_{single} \mid PI_{single} \mid \Gamma_\omega \mid Sim \mid HL \mid Opt \rangle \tag{B.1}$$

The scheduling problem - $\mathbf{SP_{single}}$

The scheduling environment contains a set of heterogeneous physical servers with various resource capacities. These machines are available in parallel to process jobs, forming a single-stage scheduling system $\mathrm{SP}_{\mathrm{single}}$. Jobs can be completed if machines can satisfy their requested computing requirements. Thus, the structure of the scheduling environment can be expressed as a parallel machine scheduling problem $\alpha = Q_m$. The following bullet points summarize the constraints and objectives:

- Release date $\beta_1 = r_j$: Jobs are released for scheduling based on their associated release time during the scheduling period T.
- Machine capacity constraint $\beta_2 = M^C$: Jobs are scheduled on physical machines only if a machine fulfills its requested computing requirements (cf. Section 2.1.2, Equation 2.1).
- The total energy consumption $\gamma_1 = E$: The energy consumption of computing resources is a significant cost and sustainability driver in a cloud environment. Therefore, scheduling problems in cloud environments are solved by minimizing overall energy consumption.
- The total number of penalties $\gamma_2 = U$: SLA violations cost loss of reputation and are associated with financial penalties. Therefore, solution methods for scheduling problems in cloud environments must be designed to take into consideration the minimization of total penalties and/or violations in SLAs.

Given the discussed cloud environment description, the scheduling problem is expressed by $SP_{single} = \langle Q_m \mid r_j,\ M^C \mid E,\ U \rangle$.

Problem instance - $\mathbf{PI_{single}}$

For evaluation, we utilized ten problem instances from workload traces of the PlanetLab benchmark. To ensure comparability, we used the same workload traces that the authors utilize in (Moges and Abebe, 2019; Beloglazov et al., 2012a). Based on the data structure presented in Section 3.4.2, the instantiation is summarized in the following bullet points:

- Let the set $T = \{T_t, \ldots, T_{|T|}\} : \forall\ t \in \{1, \ldots, 24\}$ denote a day scheduling period divided per hour. Given a discrete change in time horizon ΔT_1, we move from T_1 to T_2.
- Let the set $J = \{J_j, \ldots, J_n\} : \forall\ j \in \{1, \ldots, 1516\}$ denote the number of jobs, which must be allocated during the scheduling period T.
- Let the set $M^C = \{M_i^C, \ldots, M_m^C\} : \forall\ i \in \{1, \ldots, 560\}$ denote the number of machines that are available to process jobs in parallel.
- Let $p_{i,j} \in \mathbb{R}^+$ denote the required processing time of a job $J_j \in J$ to be hosted and processed by a machine $M_i^C \in M^C$.

Objective function - Γ_ω:

Based on the presented instantiation of the scheduling problem and the description of the problem instance, the scheduling problems are solved subject to the minimization of the objective function in Equation B.2. The problem instances are solved by finding a solution $X \in \mathbb{X}$ that minimizes the objective values. The overall objective function is formulated using a weighted sum approach such that the set $\Gamma = \{\gamma_i, \ldots, \gamma_{|\Gamma|}\} : \forall\ i \in \{1, \ldots, 3\}$ is associated with the set of weights $\omega = \{\omega_1 = 0.6,\ \omega_1 = 0.2,\ \omega_3 = 0.2\}$ as presented in Equation B.2.

$$\arg\min_{X \in \mathbb{X}} \Gamma_\omega(X) = \arg\min_{X \in \mathbb{X}} (\omega_1 \cdot \gamma_1(X) + \omega_2 \cdot \gamma_2(X) + \omega_3 \cdot \gamma_3(X)) \tag{B.2}$$

Simulation - (Sim)

CloudSim Plus was utilized to build a simulation model for modeling the defined cloud environment. The simulation model is crucial for evaluation and solution construction in the presented methodology.

Heuristic Library - (HL)

We will independently summarize the developed and adopted heuristics in the first and second use cases. The created simulation environment accesses a predefined set of scheduling allocation heuristics. These allocation heuristics are used during the scheduling period T to manage the scheduling and rescheduling of jobs to machines. The conducted experiments are compared to the scheduling heuristic presented in related works (Moges and Abebe, 2019; Beloglazov et al., 2012a; Beloglazov et al., 2012b). Nine allocation heuristics presented in these works were utilized and integrated with the optimization component. The adopted algorithms are summarized in the coming bullet points:

- Power Efficient First-Fit Decreasing (PEFFD) based on Moges and Abebe (2019).
- Power Efficient Best-Fit Decreasing (PEBFD) based on Moges and Abebe (2019).
- Medium-Fit Power Efficient Decreasing (MFPED) based on Moges and Abebe (2019).
- Modified Best-Fit Decreasing (MBFD) based on Beloglazov et al. (2012a).
- Best-Fit Static Threshold (BFTHR) based on Beloglazov et al. (2012b).
- Power-Aware Best-Fit Decreasing (PABFD) based on Beloglazov et al. (2012b).
- Median Absolute Deviation (MAD) based on Beloglazov et al. (2012b).
- Static threshold-based scheduling policy (THR) based on Beloglazov et al. (2012b).

Optimization - (Opt)

The optimization component of MESEAS was instantiated using the indirect discrete encoding model. The included allocation heuristics were used as a marker to define the genotypes for the implemented Genetic Algorithms (GA). The genotypes were grouped into a chromosome represented by a vector Φ^A as presented in Equation B.3.

$$\Phi^A = \left[\Phi_\varphi, \quad \cdots, \quad \Phi_{|T|}\right] : \quad (\varphi = 1, 2, \ldots, 24) \wedge (\Phi_\varphi = 1, 2, \ldots, 8) \tag{B.3}$$

B.2.2 Computational results

Table B.4: Achieved improvement compared to heuristic methods in terms of minimizing energy consumption.

Heuristic method	Energy consumption (kWh)	MESEAS improvement (%)
MAD	912.83	65.0
THR	912.75	65.0
MBFD	902.62	64.7
BFTHR	833.97	61.7
PEBFD	610.5	47.7
MFPED	584.26	45.4
PEFFD	573.56	44.4
PABFD	460.89	30.8
MESEAS-Single	319,06	–
Average improvement	53.1 %	

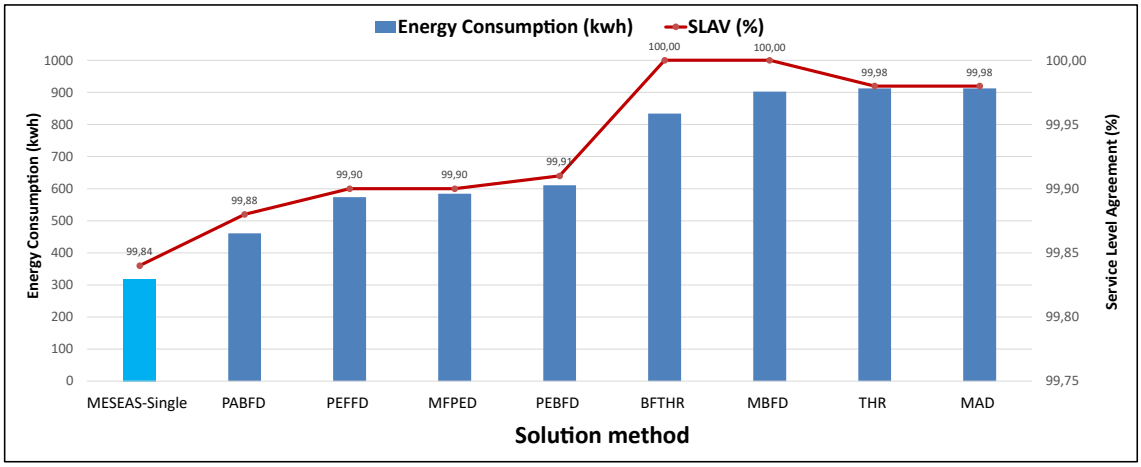


Figure B.1: Average energy consumption and SLA in percentage (Nahhas et al., 2021a).

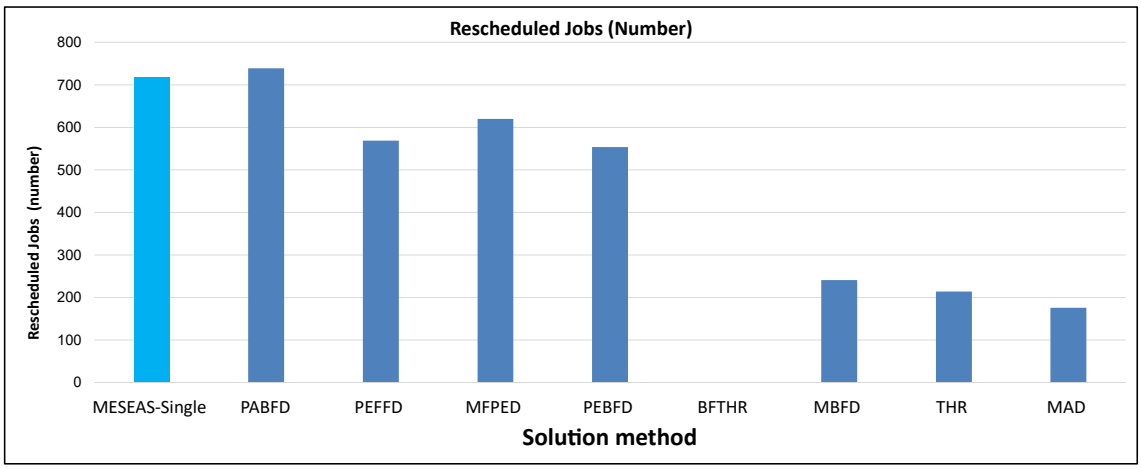


Figure B.2: Average number of rescheduled jobs (Nahhas et al., 2021a).

C Scheduling in manufacturing environments

C.1 First use case: Imitation learning

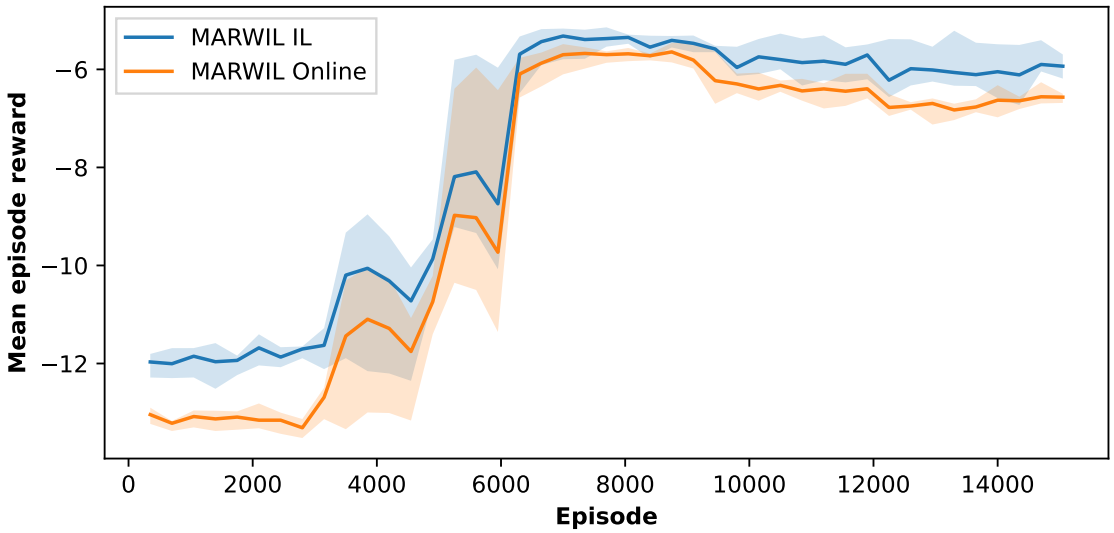


Figure C.1: Comparison between MARWIL trained with NSGA III solutions (MARWIL IL) and pure MARWIL without previous experiences (Nahhas et al., 2024a).

C.2 First use case: Performance comparison in percentages

Table C.1: Achieved improvement compared to heuristic and metheuristic methods in terms of minimizing the makespan in percentage.

PI	Multi vs. Sim (EDD) (%)	Multi vs. (GA & DA-SI) (%)	Multi vs. Single (%)
1	12.1	9.2	1.6
2	13.6	10.1	1.0
3	12.2	2.4	1.2
4	9.2	6.3	0.9
5	15.0	11.4	5.1
6	14.2	1.9	0.5
7	8.6	4.5	2.4
8	6.3	17.0	0.1
9	8.0	1.8	6.0
10	15.5	10.6	0.3
11	9.3	13.6	2.3
12	13.5	20.9	2.6
13	21.9	17.4	2.4
14	13.5	12.0	3.8
15	11.0	13.5	4.4
16	12.3	8.0	5.0
17	17.2	17.5	4.5
18	10.5	18.6	2.5
19	14.7	7.8	3.1
20	12.7	12.4	1.1
21	15.7	6.0	2.0
22	11.6	0.8	2.3
23	15.1	21.0	6.3
24	17.1	11.6	6.1
25	8.9	18.8	0.9
26	7.5	20.9	1.3
27	7.5	16.9	1.8
28	14.5	2.4	0.9
29	13.7	7.4	2.7
30	18.7	6.8	4.6
Ave	12.72	10.97	2.66

Table C.2: Achieved improvement compared to heuristic and metheuristic methods in terms of minimizing major setup times in percentage.

PI	Multi vs. Sim (EDD) (%)	Multi vs. (GA & DA-SI) (%)	Multi vs. Single (%)
1	65.2	11.5	9.8
2	70.1	4.8	0.0
3	62.2	4.0	0.0
4	60.8	8.9	-6.3
5	67.8	11.4	7.1
6	66.1	14.0	15.7
7	70.5	11.6	9.5
8	68.9	8.7	8.7
9	67.9	6.5	6.5
10	70.5	2.5	2.5
11	65.7	19.3	9.8
12	66.7	0.0	0.0
13	72.2	5.1	5.1
14	68.0	2.4	4.7
15	72.4	9.8	2.6
16	72.7	17.4	5.0
17	70.4	7.0	0.0
18	66.4	6.5	8.5
19	68.9	18.0	12.8
20	67.4	0.0	2.2
21	68.3	16.7	16.7
22	68.1	4.3	0.0
23	66.9	6.7	2.3
24	70.3	4.7	8.9
25	66.2	7.7	2.0
26	68.8	18.8	11.4
27	68.6	10.4	8.5
28	69.3	15.7	6.5
29	69.5	13.0	7.0
30	73.1	7.9	5.4
Ave	68.32	9.17	5.77

C.3 Second use case: Performance comparison to related works in percentages

Table C.3: MESEAS-Multi vs. NEAT %

Improvement	Makespan (%)	Major setup (%)	Tardiness (%)	Penalties (%)
PI-1	5.0	56.1	100	100
PI-2	4.7	65.8	100	100
PI-3	10.6	59.2	0	0
PI-4	2.0	57.5	0	0
Average	5.6	59.6	50	50
Importance	0.5	0.5	0.5	0.5
Average improvement	32.6 %		50 %	

Table C.4: MESEAS-A3C vs. NEAT %

Improvement	Makespan (%)	Major setup (%)	Tardiness (%)	Penalties (%)
PI-1	3.9	49.1	100	100
PI-2	15.5	63.1	100	100
PI-3	14.1	60.6	0	0
PI-4	-5.3	54.0	-100	-100
Average	7.0	56.7	25	25
Importance	0.5	0.5	0.5	0.5
Average improvement	31.9 %		25 %	

Table C.5: MESEAS-PPO vs. NEAT %

Improvement	Makespan (%)	Major setup (%)	Tardiness (%)	Penalties (%)
PI-1	-4.7	52.6	100	100
PI-2	14.5	63.1	100	100
PI-3	11.4	62.0	0	0
PI-4	-6.4	54.0	-100	-100
Average	3.7	57.9	25	25
Importance	0.5	0.5	0.5	0.5
Average improvement	30.8 %		25 %	

Bibliography

Aarts, E., Korst, J., and Michiels, W. (2005). Simulated annealing. In Burke, E. K. and Kendall, G. (eds.), Search Methodologies, pages 187–210. Springer US, Boston, MA. 48

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. 156

Adamuthe, A. C. and Bichkar, R. S. (2012). Tabu search for solving personnel scheduling problem. In 2012 International Conference on Communication, Information & Computing Technology (ICCICT), pages 1–6. 48

Ahmed, Z., Roux, N. L., Norouzi, M., and Schuurmans, D. (2018). Understanding the impact of entropy on policy optimization. ArXiv, abs/1811.11214. 193

Akhter, N. and Othman, M. (2016). Energy aware resource allocation of cloud data center: review and open issues. Cluster Computing, 19(3):1163–1182. 25, 41

Ali Asghar Heidari, Seyedali Mirjalili, Hossam Faris, Ibrahim Aljarah, Majdi Mafarja, and Huiling Chen (2019). Harris hawks optimization: Algorithm and applications. Future Generation Computer Systems, 97:849–872. 69

Allaoui, H. and Artiba, A. (2004). Integrating simulation and optimization to schedule a hybrid flow shop with maintenance constraints. Computers & Industrial Engineering, 47(4):431–450. 25, 48, 51

Almeida, D., Ferreira, L. P., Sá, J. C., Lopes, M., da Silva, F. J. G., and Pereira, M. (2022). Performance evaluation of dispatching rules and simulated annealing in a scheduling problem from a quality-functionality perspective. In Moldovan (ed.), 15th International Conference Interdisciplinarity in Engineering, pages 258–267. Springer International Publishing, [S.l.]. 38, 39

Alves, J. C. and Mateus, G. R. (2020). Deep reinforcement learning and optimization approach for multi-echelon supply chain with uncertain demands. In International Conference on Computational Logistics. 57, 58, 62, 76

Alwadi, A., Nahhas, A., Bosse, S., Jamous, N., and Turowski, K. (2018). Toward a performance requirements model for the early design phase of it systems. In Sixth International Conference on Enterprise Systems (ES), Piscataway, NJ. 16, 80

Alwadi, A., Nahhas, A., Bosse, S., Jamous, N., and Turowski, K. (2019). A modernized model for performance requirements and their interdependencies. In 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA). 16, 80

Amdahl, G. M., Blaauw, G. A., and Brooks, F. P. (1964). Architecture of the ibm system/360. IBM Journal of Research and Development, 8(2):87–101. 144

Amer, D. A., Attiya, G., Zeidan, I., and Nasr, A. A. (2022). Elite learning harris hawks optimizer for multi-objective task scheduling in cloud computing. Journal of Supercomputing, 78(2):2793–2818. 69, 71

Amoretti, M., Zanichelli, F., and Conte, G. (2013). Efficient autonomic cloud computing using online discrete event simulation. Journal of Parallel and Distributed Computing, 73(6):767–776. 3

Andersson, M., Ng, A. H. C., and Grimm, H. (2008). Simulation optimization for industrial scheduling using hybrid genetic representation. In Proceedings of the 40th Conference on Winter Simulation, pages 2004–2011. 50

Angerhofer, B. J. and Angelides, M. C. (2000). System dynamics modelling in supply chain management: research review. In Simulation conference, 2000. proceedings. winter, pages 342–351. 31

Anuradha, V. P. and Sumathi, D. (2014). A survey on resource allocation strategies in cloud computing. In International Conference on Information Communication and Embedded Systems (ICICES2014), pages 1–7. 52

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6):26–38. 57, 58

Arunarani, A., Manjula, D., and Sugumaran, V. (2019). Task scheduling techniques in cloud computing: A literature survey. Future Generation Computer Systems, 91:407–415. 1, 2

Ashwin, S., Shankaranarayanan, V., lamy, D., Anbuudayasankar, S. P., and Thenarasu, M. (2022). Development and analysis of efficient dispatching rules for minimizing flow time and tardiness-based performance measures in a job shop scheduling. In Reddy (ed.), Intelligent Manufacturing and Energy Sustainability, pages 337–345. Springer Singapore, [S.l.]. 39

Aurich, P., Nahhas, A., Reggelin, T., Krist, M., Bruzzone, AG, de Felice, F., Frydman, C., Longo, F., and Massei, M. (2017). Simulation based optimization of a four stage hybrid flow shop with sequence-dependent setup times and availability constraints. In Bruzzone, A., Solis, A., Massei, M., De Felice, F., Longo, F., and Frydman, C. (eds.), 16th International Conference on Modeling and Applied Simulation, MAS 2017, Held at the International Multidisciplinary Modeling and Simulation Multiconference, I3M 2017, pages 144–152. CAL-TEK S.r.l. 11, 17, 104

Aurich, P., Nahhas, A., Reggelin, T., and Tolujew, J. (2016). Simulation-based optimization for solving a hybrid flow shop scheduling problem. In Proceedings of the 2016 Winter Simulation Conference, pages 2809–2819. 10, 48, 49

Baas, J., Schotten, M., Plume, A., Côté, G., and Karimi, R. (2020). Scopus as a curated, high-quality bibliometric data source for academic research in quantitative science studies. Quantitative Science Studies, 1(1):377–386. 63

Baker, K. R. and Trietsch, D. (2009). Principles of sequencing and scheduling. Wiley-Blackwell, Oxford. 1, 2, 16, 19, 20, 21, 23, 26, 27, 28, 29, 32, 34, 37, 38, 39, 41, 106

Bandaranayake, K., Jayasena, K., and Kumara, B. T. G. S. (2020). An efficient task scheduling algorithm using total resource execution time aware algorithm in cloud computing. In 2020 IEEE International Conference on Smart Cloud (SmartCloud), Washington, DC, USA, pages 29–34. 40, 170

Beloglazov, A., Abawajy, J., and Buyya, R. (2012a). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems, 28(5):755–768. 13, 44, 45, 163, 169, 170, 223, 224

Beloglazov, A., Abawajy, J., and Buyya, R. (2012b). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems, 28(5):755–768. 44, 45, 163, 169, 224

Beloglazov, A. and Buyya, R. (2010). Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. MGC@ Middleware, 4. 30

Ben-Daya, M. and Al-Fawzan, M. (1998). A tabu search approach for the flow shop scheduling problem. European Journal of Operational Research, 109(1):88–95. 48

Bertsekas, D. P. (2005). Dynamic programming and optimal control. Athena Scientific, Belmont, MA, USA. 58

Bhattacharyya, S. (ed.) (2018). Hybrid metaheuristics: Research and applications / siddhartha bhattacharyya. World Scientific Publishing, Singapore. 3, 82

Blackstone, J. H., Phillips, D. T., and Hogg, G. L. (1982). A state-of-the-art survey of dispatching rules for manufacturing job shop operations. International Journal of Production Research, 20(1):27–45. 2, 29, 35

Boettiger, C. and Eddelbuettel, D. (2017). An introduction to rocker: Docker containers for r. 146

Borshchev, A. and Filippov, A. (2004). From system dynamics and discrete event to practical agent based modeling: Reasons, techniques, tools. In The 22nd International Conference of the System Dynamics Society. 31, 32

Bosse, S., Nahhas, A., and Turowski, K. (2020). Quantitative analysis of the effects of different carbon tax levels on emissions and costs of data centers. In WI2020 Zentrale Tracks, pages 1349–1363. GITO Verlag. 16

Botoeva, E., Calvanese, D., Cogrel, B., and Xiao, G. (2017). Expressivity and complexity of mongodb (extended version). 158

Brah, S. A. and Hunsucker, J. L. (1991). Branch and bound algorithm for the flow shop with multiple processors. European Journal of Operational Research, 51(1):88–99. 29

Branke, J., Hildebrandt, T., and Scholz-Reiter, B. (2015). Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. Evolutionary computation, 23(2):249–277. 83, 86

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. 155

Brucker, P. (2007). Scheduling algorithms. Springer, Berlin and New York. 25, 28

Buduma, N. and Locascio, N. (2017). Fundamentals of deep learning: Designing next-generation machine intelligence algorithms. O'Reilly Media. 56, 59, 155

Burks, A. W., Goldstine, H. H., and von Neumann, J. (1947). Preliminary discussion of the logical design of an electronic computing instrument. Institute for Advanced Study (IAS), Princeton, NJ. 144

Burks, A. W., Goldstine, H. H., and von Neumann, J. (1982). Preliminary discussion of the logical design of an electronic computing instrument. In Randell, B. (ed.), The Origins of Digital Computers: Selected Papers, pages 399–413. Springer Berlin Heidelberg, Berlin, Heidelberg. 144

Buyya, R. and Murshed, M. (2002). Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and Computation: Practice and Experience, 14(13-15):1175–1220. 151

Buyya, R., Ranjan, R., and Calheiros, R. N. (2009). Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In Smari, W. W. (ed.), 2009 International Conference on High Performance Computing & Simulation, Piscataway, NJ, pages 1–11. 44, 151

Calheiros, R. N., Ranjan, R., Beloglazov, A., de Rose, C. A. F., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience, 41(1):23–50. 151

Cals, B., Zhang, Y., Dijkman, R., and van Dorst, C. (2021). Solving the online batching problem using deep reinforcement learning. Computers and; Industrial Engineering, 156:107221. 57, 60

Campos Ciro, G., Dugardin, F., Yalaoui, F., and Kelly, R. (2016). A nsga-ii and nsga-iii comparison for solving an open shop scheduling problem with resource constraints. IFAC-PapersOnLine, 49(12):1272–1277. 54

Caviglione, L., Gaggero, M., Paolucci, M., and Ronco, R. (2021). Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters. Soft Computing, 25(19):12569–12588. 70, 71

Challita, S., Paraiso, F., and Merle, P. (2017). A study of virtual machine placement optimization in data centers. In 7th International Conference on Cloud Computing and Services Science (CLOSER), pages 343–350. 2, 12, 30

Chandrasekaran, K. and Divakarla, U. (2013). Load balancing of virtual machine resources in cloud using genetic algorithm. In ICCN conference at National Institute of Technology Karnataka, Surathkal, pages 156–168. 52

Chang, J., Yu, D., Zhou, Z., He, W., and Zhang, L. (2022). Hierarchical reinforcement learning for multi-objective real-time flexible scheduling in a smart shop floor. Machines, 10(12). 74, 75

Chankong, V. and Haimes, Y. Y. (2008). Multiobjective decision making: theory and methodology. Courier Dover Publications. 53

Chen, J., Wang, Y., and Liu, T. (2021). A proactive resource allocation method based on adaptive prediction of resource requests in cloud computing. Eurasip Journal on Wireless Communications and Networking, 2021(1). 30, 68, 71, 77, 83

Chen, S., Wang, J., Yan, M., Yang, C., and Han, H. (2022). Research on multi-factory combination optimization based on dostar. Array, 15. 73, 75

Cheng, R., Gen, M., and Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling

problems using genetic algorithms—i. representation. Computers & Industrial Engineering, 30(4):983–997. 50

Cheng, Y. and Xu, G. (2019). A novel task provisioning approach fusing reinforcement learning for big data. IEEE Access, 7:143699–143709. 70, 71

Chernigovskaya, M., Nahhas, A., Kharitonov, A., and Turowski, K. (2024). Hyperparameter optimization in the context of smart manufacturing: a systematic literature review. Procedia Computer Science, 232:804–812. 5th International Conference on Industry 4.0 and Smart Manufacturing (ISM 2023). 211

Cheyyanda, J. T. (2020). Energy-aware genetic algorithm for load management under performance constraints in cloud datacenters. Msc thesis, Faculty of Computer Science, Otto-von-Guericke-Universität Magdeburg, Germany, Magdeburg. Assessed and supervised by Turowski, K., Krull, C., and Nahhas, A. 169

Chica, M., Juan, A. A., Bayliss, C., Cordón, O., and W. David Kelton, O. (2020). Why simheuristics? benefits, limitations, and best practices when combining metaheuristics with simulation. SORT. Statistics and Operations Research Transactions, 44(2):311–334. 4

Chou, Y.-L., Yang, J.-M., and Wu, C.-H. (2020). An energy-aware scheduling algorithm under maximum power consumption constraints. Journal of Manufacturing Systems, 57:182–197. 30

Conway, R. W., Maxwell, W. L., and Miller, L. W. (1967). Theory of scheduling. Addison-Wesley Pub. Co. 9, 16, 26, 29

Cook, S. A. (1971). The complexity of theorem-proving procedures. In Harrison, M. A., Banerji, R. B., and Ullman, J. D. (eds.), the third annual ACM symposium, pages 151–158. 9, 33

Crainic, T. G. and Toulouse, M. (2003). Parallel strategies for meta-heuristics. In Glover, F. and Kochenberger, G. A. (eds.), Handbook of Metaheuristics, pages 475–513. Springer Science & Business Media. 48

Crowston, W. B., Glover, F., thompson, G. L., and Trawick, J. D. (1963). Probabilistic and parametric learning combinations of local job shop scheduling rules. Defense Technical Information Center, Fort Belvoir, VA. 3, 82

Csaszar, P., Tirpak, T. M., and Nelson, P. C. (2000). Optimization of a high-speed placement machine using tabu search algorithms. Annals of Operations Research, 96(1):125–147. 171

Cui, L. and Tang, Y. (2023). Comparing the effectiveness of ppo and its variants in

training ai to play game. In 2023 International Conference on Cyber-Physical Social Intelligence (ICCSI), pages 521–526. 60

Daase, C., Haertel, C., Nahhas, A., and Turowski, K. (2024). Classifying design science research in terms of types of reasoning from an epistemological perspective. In Mandviwalla, M., Söllner, M., and Tuunanen, T. (eds.), Design Science Research for a Resilient Future, Cham, pages 155–167. 8

Daase, C., Haertel, C., Nahhas, A., Volk, M., Steigerwald, H., Ramesohl, A., Schneider, B., Zeier, A., and Turowski, K. (2023). Following the digital thread – a cloud-based observation. Procedia Computer Science, 217:1867–1876. 4th International Conference on Industry 4.0 and Smart Manufacturing. 10

Dai, Y., Wang, G., Muhammad, K., and Liu, S. (2022). A closed-loop healthcare processing approach based on deep reinforcement learning. Multimedia Tools and Applications, 81. 76

Das, I. and Dennis, J. E. (1998). Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. SIAM Journal on Optimization, 8(3):631–657. 54, 179

de Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. University of Michigan. 167, 179

Deb, K. and Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. IEEE Transactions on Evolutionary Computation, 18(4):577–601. 53, 54, 126, 178, 179

del Real Torres, A., Andreiana, D. S., Ojeda Roldán, A., Hernández Bustos, A., and Acevedo Galicia, L. E. (2022). A review of deep reinforcement learning approaches for smart manufacturing in industry 4.0 and 5.0 framework. Applied Sciences, 12(23). 57, 62, 76

Delalleau, O., Peter, M., Alonso, E., and Logut, A. (2019). Discrete and continuous action representation for practical rl in video games. URL https://arxiv.org/pdf/1912.11077. 178

Dey, S., De, S., and Bhattacharyya, S. (2018). Introduction to hybrid metaheuristics. In Bhattacharyya, S. (ed.), Hybrid metaheuristics, pages 1–38. World Scientific Publishing, Singapore. 3, 82

Dobbelaere, P. and Esmaili, K. S. (2017). Kafka versus rabbitmq. 158

Dror G. Feitelson, Dan Tsafrir, and David Krakov (2014). Experience with using the

parallel workloads archive. Journal of Parallel and Distributed Computing, 74(10):2967–2982. 14, 169

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. 62, 76

Ebben, M. J. R., Hans, E. W., and Olde Weghuis, F. M. (2005). Workload based order acceptance in job shop environments. OR Spectrum, 27(1):107–122. 23

Engelmann, T., Kharitonov, A., Nahhas, A., Staegemann, D. G., Haertel, C., Daase, C., and Turowski, K. (2024). Self-hosted open-source dependency management solutions. In Yang, X.-S., Sherratt, S., Dey, N., and Joshi, A. (eds.), Proceedings of Ninth International Congress on Information and Communication Technology, Singapore, pages 133–148. 146

Eshelman, L. J., Caruana, R. A., and Schaffer, J. D. (1989). Biases in the crossover landscape. In Schaffer, J. D. (ed.), Proceedings of the third international conference on Genetic algorithms, San Francisco, CA, USA, pages 10–19. 167, 179

Ferreira, J. S. (2013). Multimethodology in metaheuristics. Journal of the Operational Research Society, 64(6):873–883. 82

Forrester, J. W. (1968). Industrial dynamics—after the first decade. Management Science, 14(7):398–415. 31

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research, 13:2171–2175. 155

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. 59

Gao, Y. and Wang, Y. (2022). Multiple workflows offloading based on deep reinforcement learning in mobile edge computing. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 13155 LNCS:476–493. 70, 71

Geem, Z. W., Kim, J. H., and Loganathan, G. V. (2001). A new heuristic optimization algorithm: Harmony search. SIMULATION, 76(2):60–68. 50

Ghafari, R., Kabutarkhani, F. H., and Mansouri, N. (2022). Task scheduling algorithms for energy optimization in cloud environment: a comprehensive review. Cluster Computing, 25(2):1035–1093. 5, 30, 35, 41, 82

Gilmore, P. C. and Gomory, R. E. (1964). Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. Operations Research, 12(5):655–679.

33

Glover, F. (1989). Tabu search—part i. ORSA Journal on computing, 1(3):190–206. 10, 49

Glover, F. and Kochenberger, G. A. (eds.) (2003). Handbook of metaheuristics. Springer Science & Business Media. 48

Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, Mass. and Wokingham. 11, 49, 53

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. Foundations of genetic algorithms, 1:69–93. 51

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. MIT Press. 57

Gordon, G. (1961). A general purpose systems simulation program. In Ware, W. H. (ed.), Eastern joint computer conference: computers - Key to total systems control, New York, NY, USA, pages 87–104. 32

Goudarzi, P., Hosseinpour, M., Goudarzi, R., and Lloret, J. (2022). Holistic utility satisfaction in cloud data centre network using reinforcement learning. Future Internet, 14(12). 69, 71

Govindaiah, S. and Pey, M. D. (2019). Applying reinforcement learning to plan manufacturing material handling part 1: Background and formal problem specification. ACMSE 2019 - Proceedings of the 2019 ACM Southeast Conference. 72, 75

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E. J. and B.H. Korte (eds.), Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver, pages 287–326. Elsevier. 16, 19, 20, 24, 26, 31, 96, 172

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. Nature, 538(7626):471–476. 57

Großmann, A. and Poli, R. (1999). Learning a navigation task in changing environments by multi-task reinforcement learning. In European Workshop on Learning Robots, pages 23–43. 56

Gu, Z., Jia, Z., and Choset, H. (2019). Adversary a3c for robust reinforcement learning. 61

Guevara, J. C., Torres, R., Bittencourt, L. F., and Da Fonseca, N. (2022). Qos-aware task scheduling based on reinforcement learning for the cloud-fog continuum. 2022 IEEE Global Communications Conference, GLOBECOM 2022 - Proceedings. 68, 71

Gupta, J. N. and Stafford, E. F. (2006). Flowshop scheduling research after five decades. European Journal of Operational Research, 169(3):699–711. 2, 33, 45

Gupta, J. N. D. (1988). Two-stage, hybrid flowshop scheduling problem. The Journal of the Operational Research Society, 39(4):359. 34, 46, 47, 51

Gupta, S. K. and Kyparisis, J. (1987). Single machine scheduling research. Omega, 15(3):207–227. 24, 33, 34

Gustineli, M. (2022). A survey on recently proposed activation functions for deep learning. 57

Ham, R. (2018). salabim: discrete event simulation and animation in python. Journal of Open Source Software, 3:767. 152

Hammami, N. E. H., Lardeux, B., Hadj-Alouane, A. B., and Jridi, M. (2022). Job shop scheduling: A novel drl approach for continuous schedule-generation facing real-time job arrivals. IFAC-PapersOnLine, 55(10):2493–2498. 10th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2022. 57

Hansen, N. (2023). The cma evolution strategy: A tutorial. 58

He, J., Chen, J., He, X., Gao, J., Li, L., Deng, L., and Ostendorf, M. (2016). Deep reinforcement learning with a natural language action space. 76

He, Z., Wang, K., Li, H., Song, H., Lin, Z., Gao, K., and Sadollah, A. (2022). Improved q-learning algorithm for solving permutation flow shop scheduling problems. IET Collaborative Intelligent Manufacturing, 4(1):35–44. 72, 75

Heinrich, J. and Silver, D. (2016). Deep reinforcement learning from self-play in imperfect-information games. 76

Helal, M., Rabelo, L., Sepúlveda, J., and Jones, A. (op. 2007). A methodology for integrating and synchronizing the system dynamics and discrete event simulation paradigms. In Sterman, J. D. (ed.), Proceedings of the 25th International Conference [of the] System Dynamics Society and 50th Anniversary Celebration, Boston, Massachusetts, USA, July 29 - August 2, 2007, pages 1–24. The System Dynamics Society, Albany (New York). 32

Held, M. and Karp, R. M. (1961). A dynamic programming approach to sequencing problems. In Proceedings of the 1961 16th ACM National Meeting, New York, NY, USA, pages 196–210. 34

Holland, J. H. (1975). Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence / by john h.holland. University of Michigan Press, Ann Arbor, Mich. 3, 11, 49, 126, 166, 179

Houssein, E. H., Gad, A. G., Wazery, Y. M., and Suganthan, P. N. (2021). Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends. Swarm and Evolutionary Computation, 62:100841. 81

Howell, F. and McNab, R. (1998). Simjava: A discrete event simulation library for java. Simulation Series, 30:51–56. 151

Hunsucker, J. L. and Shah, J. R. (1994). Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment. European Journal of Operational Research, 72(1):102–114. 35, 38

Iivari, J. (2007). A paradigmatic analysis of information systems as a design science. Scandinavian Journal of Information Systems, 19(2). 8

ISO/IEC 19505-2: 2012, Information technology (2012). Object Management Group Unified Modeling Language (OMG UML). International Organization for Standardization. 91

Jalalian, Z. and Sharifi, M. (2022). A hierarchical multi-objective task scheduling approach for fast big data processing. Journal of Supercomputing, 78(2):2307–2336. 68, 71

Jayanetti, A., Halgamuge, S., and Buyya, R. (2022). Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments. Future Generation Computer Systems, 137:14–30. 24, 30

Jiang, F., Ferriter, K., and Castillo, C. (2020). A cloud-agnostic framework to enable cost-aware scheduling of applications in a multi-cloud environment. In NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, [S.l.], pages 1–9. 1

Johnson, D. S. (2012). A brief history of np-completeness, 1954-2012. Documenta Mathematica, Extra Volume ISMP:359–376. 33

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly, 1(1):61–68. 34, 37, 47

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. Artificial Intelligence, 101(1-2):99–134. 56

Kaiser, S., Haq, M. S., Tosun, A. S., and Korkmaz, T. (2022). Container technologies for arm architecture: A comprehensive survey of the state-of-the-art. IEEE Access, 10:84853–84881. 147

Kalczynski, P. J. and Kamburowski, J. (2007). On the neh heuristic for minimizing the makespan in permutation flow shops. Omega, 35(1):53–60. 51

Kanervisto, A., Scheller, C., and Hautamaki, V. (2020). Action space shaping in deep reinforcement learning. In 2020 IEEE Conference on Games (CoG), pages 479–486. 134, 178

Kanoje, S., Powar, V., and Mukhopadhyay, D. (2015). Using mongodb for social networking website. 158

Kasper, T. A., Land, M. J., and Teunter, R. H. (2023). Towards system state dispatching in high-variety manufacturing. Omega, 114:102726. 39

Katal, A., Dahiya, S., and Choudhury, T. (2023). Energy efficiency in cloud computing data centers: a survey on software technologies. Cluster Computing, 26(3):1845–1875. 12, 16

Kharitonov, A., Nahhas, A., Müller, H., and Turowski, K. (2023). Data driven meta-heuristic-assisted approach for placement of standard it enterprise systems in hybrid-cloud. In Proceedings of the 13th International Conference on Cloud Computing and Services Science. 54

Kharitonov, A., Nahhas, A., Müller, H., and Turowski, K. (2024). Towards hybrid-cloud infrastructure composition for sap systems landscapes in smart manufacturing. Procedia Computer Science, 232:2376–2385. 5th International Conference on Industry 4.0 and Smart Manufacturing (ISM 2023). 11

Kharitonov, A., Nahhas, A., Pohl, M., and Turowski, K. (2022). Comparative analysis of machine learning models for anomaly detection in manufacturing. Procedia Computer Science, 200:1288–1297. 17, 197

Kianpour, P., Gupta, D., Krishnan, K., and Gopalakrishnan, B. (2021). Optimising unrelated parallel machine scheduling in job shops with maximum allowable tardiness limit. International Journal of Industrial and Systems Engineering, 37(3):359. 27

Kim, J. and Yoo, S. (2019). Software review: Deap (distributed evolutionary algorithm in python) library. Genetic Programming and Evolvable Machines, 20(1):139–142. 155

Kim, N., Barde, S., Bae, K., and Shin, H. (2023). Learning per-machine linear dispatching rule for heterogeneous multi-machines control. International Journal of Production Research, 61(1):162–182. 84

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. Science (New York, N.Y.), 220(4598):671–680. 3, 10, 48

Kis, T. and Pesch, E. (2005). A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. European Journal of Operational Research, 164(3):592–608. 34

Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004, pages 2619–2624 Vol.3. 57

Konda, V. and Tsitsiklis, J. (2000). Onactor-critic algorithms. SIAM Journal on Control and Optimization, 42. 59

Koomey, J. (2011). Growth in data center electricity use 2005 to 2010. A report by Analytical Press, completed at the request of The New York Times, 9. 16, 30

Koomey, J. G. (2008). Worldwide electricity used in data centers. Environmental Research Letters, 3(3):034008. 30

Koulamas, C. (1994). The total tardiness problem: Review and extensions. Operations Research, 42(6):1025–1041. 2

Koutsoukas, A., Monaghan, K. J., Li, X., and Huan, J. (2017). Deep-learning: investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data. Journal of Cheminformatics, 9(1):42. 193

Krahl, D. (2007). Extendsim 7. In 2007 Winter Simulation Conference, pages 226–232. 151

Krahl, D. (2008). Extendsim 7. In Proceedings of the 40th Conference on Winter Simulation, pages 215–221. 32

Krist, M. (2022). Echtzeitfähige fertigungsfeinplanung einer hybrid-flow-shop - produktion. Dissertation, Otto-von-Guericke-Universität Magdeburg, Germany, Magdeburg. 171

Kruekaew, B. and Kimpan, W. (2022). Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning. IEEE Access, 10:17803–17818. 2, 69, 71

Kut C. So (1990). Some heuristics for scheduling jobs on parallel machines with setups. Management Science, 36(4):467–475. 24, 30

Lakhan, A., Mohammed, M., Nedoma, J., Martinek, R., Tiwari, P., and Kumar, N. (2023). Drlbts: deep reinforcement learning-aware blockchain-based healthcare system. Scientific Reports, 13:4124. 76

Lang, S., Reggelin, T., Behrendt, F., and Nahhas, A. (2020). Evolving neural networks to solve a two-stage hybrid flow shop scheduling problem with family setup times. In Bui, T. (ed.), Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS 2020), pages 1298–1307. 14, 197, 198

Lang, S., Reggelin, T., Müller, M., and Nahhas, A. (2021a). Open-source discrete-event simulation software for applications in production and logistics: An alternative to commercial tools? Procedia Computer Science, 180:978–987. 14, 17, 101

Lang, S., Reggelin, T., Schmidt, J., Müller, M., and Nahhas, A. (2021b). Neuroevolution of augmenting topologies for solving a two-stage hybrid flow shop scheduling problem: A comparison of different solution strategies. Expert Systems with Applications, 172:114666. 15, 137, 197, 208

Lee, J. and Sutton, R. S. (2021). Policy iterations for reinforcement learning problems in continuous time and space — fundamental theory and methods. Automatica, 126:109421. 58

Lee, W.-J., Kim, B.-H., Ko, K., and Shin, H. (2019). Simulation based multi-objective fab scheduling by using reinforcement learning. Proceedings - Winter Simulation Conference, 2019-December. 72, 75

Leng, J., Wang, X., Wu, S., Jin, C., Tang, M., Liu, R., Vogl, A., and Liu, H. (2022). A multi-objective reinforcement learning approach for resequencing scheduling problems in automotive manufacturing systems. International Journal of Production Research. 73, 75

Lenstra, J. K., Rinnooy Kan, A., and Brucker, P. (1977). Complexity of machine scheduling problems. In P.L. Hammer, E.L. Johnson, B.H. Korte and G.L. Nemhauser (eds.), Studies in Integer Programming, pages 343–362. Elsevier. 2, 24, 33

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. 57

Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J., and Jurafsky, D. (2016). Deep reinforcement learning for dialogue generation. 76

Li, S. (1997). A hybrid two-stage flowshop with part family, batch production, major and minor set-ups. European Journal of Operational Research, 102(1):142–156. 47

Li, Y., Sycara, K., and Iyer, R. (2018). Object-sensitive deep reinforcement learning. 62, 76

Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I. (2018). Rllib: Abstractions for distributed reinforcement

learning. 156

Liao, C.-J., Tseng, C.-T., and Luarn, P. (2007). A discrete version of particle swarm optimization for flowshop scheduling problems. Computers & Operations Research, 34(10):3099–3111. 3

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2019). Continuous control with deep reinforcement learning. 59

Lin, Y., Qu, T., Lu, Z., Su, Y., and Wei, Y. (2022). Asynchronous reinforcement learning framework and knowledge transfer for net-order exploration in detailed routing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41(9):3132–3142. 61

Liu, Q., Wang, C., Li, X., and Gao, L. (2023). A multi-population co-evolutionary algorithm for green integrated process planning and scheduling considering logistics system. Engineering Applications of Artificial Intelligence, 126:107030. 177

Lopez-Pires, F. and Baran, B. (2015). Virtual machine placement literature review. arXiv preprint arXiv:1506.01509. 25

Luo, S., Zhang, L., and Fan, Y. (2022). Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning. IEEE Transactions on Automation Science and Engineering, 19(4):3020–3038. 73, 75

Maccarthy, B. L. and Liu, J. (1993). Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. International Journal of Production Research, 31(1):59–79. 3, 33, 35, 79

March, S. T. and Smith, G. F. (1995). Design and natural science research on information technology. Decision Support Systems, 15(4):251–266. 139, 140

März, L., Krug, W., Rose, O., and Weigert, G. (2011). Simulation und optimierung in produktion und logistik: Praxisorientierter leitfaden mit fallbeispielen. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg. 32

Menaka, M. and Sendhil Kumar, K. S. (2022). Workflow scheduling in cloud environment – challenges, tools, limitations & methodologies: A review. Measurement: Sensors, 24:100436. 2

Méndez-Hernández, B. M., Rodríguez-Bazan, E. D., Martinez-Jimenez, Y., Libin, P., and Nowé, A. (2019). A multi-objective reinforcement learning algorithm for jssp. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11727 LNCS:567–584. 71, 75

Merkel, D. (2014). Docker: Lightweight linux containers for consistent development and

deployment. Linux J., 2014(239). 146

Miikkulainen, R. and Stanley, K. O. (2009). Evolving neural networks. In Rothlauf, F. (ed.), Proceedings of the 11th Annual conference on Genetic and evolutionary computation, New York, NY, USA, pages 2977–3014. 197

Mirjalili, S., Mirjalili, S. M., and Lewis, A. (2014). Grey wolf optimizer. Advances in Engineering Software, 69:46–61. 72

Mirsanei, H. S., Zandieh, M., Moayed, M. J., and Khabbazi, M. R. (2011). A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times. Journal of Intelligent Manufacturing, 22(6):965–978. 48, 49

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Balcan, M. F. and Weinberger, K. Q. (eds.), Proceedings of The 33rd International Conference on Machine Learning, New York, New York, USA, pages 1928–1937. 57, 59, 60, 61, 136

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. 57, 76

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540):529–533. 62, 76

Moges, F. F. and Abebe, S. L. (2019). Energy-aware vm placement algorithms for the openstack neat consolidation framework. Journal of Cloud Computing, 8(1). 2, 13, 44, 45, 168, 169, 170, 223, 224

Mouftah, H. T. and Kantarci, B. (2013). Energy-efficient cloud computing: A green migration of traditional it. In Obaidat, M. S., Anpalagan, A., and Woungang, I. (eds.), Handbook of green information and communication systems, pages 295–330. Elsevier/Academic Press, Amsterdam and Boston. 30

Müller, H., Kharitonov, A., Nahhas, A., Bosse, S., and Turowski, K. (2022). Addressing it capacity management concerns using machine learning techniques. SN Computer Science, 3(1). 17

Murad, S. A., Azmi, Z. R. M., Brishti, F. J., Saib, M., and Bairagi, A. K. (2023). Priority based fair scheduling: Enhancing efficiency in cloud job distribution. In 2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS), pages 170–175. 40, 81

Murad, S. A., Azmi, Z. R. M., Muzahid, A. J. M., and Al-Imran, M. (2021). Comparative study on job scheduling using priority rule and machine learning. In 2021 Emerging Technology in Computing, Communication and Electronics (ETCCE), pages 1–8. 40

Murad, S. A., Azmi, Z. R. M., Muzahid, A. J. M., Bhuiyan, M. K. B., Saib, M., Rahimi, N., Prottasha, N. J., and Bairagi, A. K. (2024). Sg-pbfs: Shortest gap-priority based fair scheduling technique for job scheduling in cloud environment. Future Generation Computer Systems, 150:232–242. 2

Nahhas, A., Aurich, P., Bosse, S., Reggelin, T., and Turowski, K. (2017a). Metaheuristic and hybrid simulation-based optimization for solving scheduling problems with major and minor setup times. In Bruzzone, A., Solis, A., Massei, M., De Felice, F., Longo, F., and Frydman, C. (eds.), 16th International Conference on Modeling and Applied Simulation, MAS 2017, Held at the International Multidisciplinary Modeling and Simulation Multiconference, I3M 2017. CAL-TEK S.r.l. 1, 3, 10, 11, 13, 17, 23, 104, 121, 136, 151

Nahhas, A., Aurich, P., Reggelin, T., and Tolujew, J. (2016). Heuristic and metaheuristic simulation-based optimization for solving a hybrid flow shop scheduling problem. In G. Bruzzone, A., De Felice, F., Frydman, C., Massei, M., Merkuryev, Y., and Solis, A. (eds.), The 15th International Conference on Modeling and Applied Simulation, pages 95–103. CAL-TEK S.r.l., RENDE (CS), ITALY. 10

Nahhas, A., Awaldi, A., and Reggelin, T. (2017b). Simulation and the emergency department overcrowding problem. Procedia Engineering, 178:368–376. 1, 10

Nahhas, A., Bosse, S., Pohl, M., and Turowski, K. (2019a). Toward an autonomic and adaptive load management strategy for reducing energy consumption under performance constraints in data centers. In Proceedings of the 9th International Conference on Cloud Computing and Services Science, pages 471–478. 5, 13, 17, 21, 30, 83, 104, 120, 136, 151, 163

Nahhas, A., Bosse, S., Staegemann, D., Volk, M., and Turowski, K. (2019b). A holistic view of the server consolidation and virtual machines placement problems. In 2019 15th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), pages 327–334. 11, 12, 121

Nahhas, A., Bosse, S., and Turowski, K. (2018a). Load distribution strategies for a sustainable it resources management. In Drews, P., Funk, B., Niemeyer, P., and Xie, L. (eds.), Multikonferenz Wirtschaftsinformatik 2018. Leuphana Universität Lüneburg Institut für Wirtschaftsinformatik, Lüneburg. 10, 11, 13, 21, 83, 104, 120, 136, 151, 163

Nahhas, A., Cheyyanda, J. T., and Turowski, K. (2021a). An adaptive scheduling framework for the dynamic virtual machines placement to reduce energy consumption in cloud

data centers. In Bui, T. (ed.), Proceedings of the 54th Hawaii International Conference on System Sciences. 1, 2, 13, 21, 24, 83, 136, 152, 169, 170

Nahhas, A., Haertel, C., Daase, C., Volk, M., Ramesohl, A., Steigerwald, H., Zeier, A., and Turowski, K. (2023a). On the integration of google cloud and sap hana for adaptive supply chain in retailing. Procedia Computer Science, 217:1857–1866. 10

Nahhas, A., Haertel, C., Daase, C., Volk, M., Ramesohl, A., Steigerwald, H., Zeier, A., and Turowski, K. (2023b). On the integration of google cloud and sap hana for adaptive supply chain in retailing. Procedia Computer Science, 217:1857–1866. 4th International Conference on Industry 4.0 and Smart Manufacturing. 207

Nahhas, A., Kharitonov, A., Alwadi, A., and Turowski, K. (2022a). Hybrid approach for solving multi-objective hybrid flow shop scheduling problems with family setup times. Procedia Computer Science, 200:1685–1694. 14, 15, 23, 24, 54, 126, 152, 186, 187, 188

Nahhas, A., Kharitonov, A., Haertel, C., and Turowski, K. (2024a). Imitation learning based on deep reinforcement learning for solving scheduling problems. In Tung X. Bui (ed.), 57th Hawaii International Conference on System Sciences, HICSS 2024, Hilton Hawaiian Village Waikiki Beach Resort, Hawaii, USA, January 3-6, 2024, pages 1649–1658. 15, 152, 188, 211

Nahhas, A., Kharitonov, A., and Turowski, K. (2022b). Deep reinforcement learning techniques for solving hybrid flow shop scheduling problems: Proximal policy optimization (ppo) and asynchronous advantage actor-critic (a3c). In Bui, T. (ed.), Proceedings of the 55th Hawaii International Conference on System Sciences. 14, 15, 23, 136, 152, 188, 191, 192, 195, 196

Nahhas, A., Kharitonov, A., and Turowski, K. (2024b). Deep reinforcement learning for solving allocation problems in supply chain: An image-based observation space. Procedia Computer Science, 232:2570–2579. 15, 207

Nahhas, A., Krist, M., and Turowski, K. (2021b). An adaptive scheduling framework for solving multi-objective hybrid flow shop scheduling problems. In Bui, T. (ed.), Proceedings of the 54th Hawaii International Conference on System Sciences, pages 1645–1654. 2, 13, 15, 23, 83, 136, 152, 172, 188

Nahhas, A., Lang, S., Bosse, S., and Turowski, K. (2018b). Toward adaptive manufacturing: Scheduling problems in the context of industry 4.0. In Sixth International Conference on Enterprise Systems (ES), Piscataway, NJ, pages 108–115. 11, 13, 17, 25, 121, 136, 151

Nawaz, M., Enscore, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega, 11(1):91–95. 51, 73

Neufeld, J. S., Gupta, J. N., and Buscher, U. (2016). A comprehensive review of flowshop group scheduling literature. Computers & Operations Research, 70:56–74. 2, 4, 5, 28, 29, 77, 81, 82

Neufeld, J. S., Schulz, S., and Buscher, U. (2023). A systematic review of multi-objective hybrid flow shop scheduling. European Journal of Operational Research, 309(1):1–23. 2, 3, 4, 30, 45, 64, 81

Newell, A. and Simon, H. A. (1975). Computer science as empirical inquiry: symbols and search. In ACM Turing Award Lectures, pages 113–126. Association of Computing Machinery, New York. 8

Niu, Q., Zhou, T., Fei, M., and Wang, B. (2012). An efficient quantum immune algorithm to minimize mean flow time for hybrid flow shop problems. Mathematics and Computers in Simulation, 84:1–25. 29

Omotehinwa, T. O. (2022). Examining the developments in scheduling algorithms research: A bibliometric approach. Heliyon, 8(5):e09510. 64, 66

Orlikowski, W. J. and Barley, S. R. (2001). Technology and institutions: What can research on information technology and research on organizations learn from each other? MIS quarterly, 25(2):145. 7

Orlin, J. B., Punnen, A. P., and Schulz, A. S. (2004). Approximate local search in combinatorial optimization. In Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, PA, USA, pages 587–596. 46

Osaba, E., Villar-Rodriguez, E., Del Ser, J., Nebro, A. J., Molina, D., LaTorre, A., Suganthan, P. N., Coello Coello, C. A., and Herrera, F. (2021). A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems. Swarm and Evolutionary Computation, 64:100888. 129

Oukil, A. and El-Bouri, A. (2021). Ranking dispatching rules in multi-objective dynamic flow shop scheduling: a multi-faceted perspective. International Journal of Production Research, 59(2):388–411. 2

Oĝuz, C. and Ercan, M. F. (2005). A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. Journal of Scheduling, 8(4):323–351. 52

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987a). The complexity of markov decision processes. Mathematics of Operations Research, 12(3):441–450. 4, 5, 129

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987b). The complexity of markov decision processes. Math. Oper. Res., 12(3):441–450. 56

Park, K. and Pai, V. S. (2006). Comon: a mostly-scalable monitoring system for planetlab.

ACM SIGOPS Operating Systems Review, 40(1):65–74. 45, 169

Patterson, D. (2018). 50 years of computer architecture: From the mainframe cpu to the domain-specific tpu and the open risc-v instruction set. In 2018 IEEE International Solid-State Circuits Conference (ISSCC 2018), Piscataway, NJ, pages 27–31. 144

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A design science research methodology for information systems research. Journal of Management Information Systems, 24(3):45–77. 140

Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. Neural Networks, 21(4):682–697. Robotics and Neuroscience. 58

Phipps, T. E. (1956). Machine repair as a priority waiting-line problem. Operations Research, 4(1):76–85. 36

Picek, S. and Golub, M. (2010). Comparison of a crossover operator in binary-coded genetic algorithms. WSEAS transactions on computers, 9:1064–1073. 51

Pierreval, H., Bruniaux, R., and Caux, C. (2007). A continuous simulation approach for supply chains in the automotive industry. Simulation Modelling Practice and Theory, 15(2):185–198. Modeling and Simulation of Manufacturing Systems and Extended Enterprises. 32

Pimminger, S., Wagner, S., Kurschl, W., and Heinzelreiter, J. (2014). Optimization as a service: On the use of cloud computing for metaheuristic optimization. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Moreno-Díaz, R., Pichler, F., and Quesada-Arencibia, A. (eds.), Computer aided systems theory - Eurocast 2013, pages 348–355. Springer, New York. 83

Pinedo, M. L. (2012). Scheduling: Theory, algorithms, and systems. Springer New York. 1, 2, 19, 20, 22, 23, 24, 25, 26, 27, 28, 32, 45, 46, 48, 49, 50

Pires, F. L. and Barán, B. (2015). A virtual machine placement taxonomy. In 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pages 159–168. 1, 2, 3, 4, 5, 16, 25, 30, 35, 64, 66, 81, 82

Pirlot, M. (1996). General local search methods. European Journal of Operational Research, 92(3):493–511. 46, 49

Qin, Y., Wang, H., Yi, S., Li, X., and Zhai, L. (2021). A multi-objective reinforcement learning algorithm for deadline constrained scientific workflow scheduling in clouds. Frontiers of Computer Science, 15(5). 68, 71

Rashida, S. Y., Sabaei, M., Ebadzadeh, M. M., and Rahmani, A. M. (2020). A memetic grouping genetic algorithm for cost efficient vm placement in multi-cloud environment. Cluster Computing, 23(2):797–836. 3

Reeves, C. (2003). Genetic algorithms. In Glover, F. and Kochenberger, G. A. (eds.), Handbook of Metaheuristics, pages 55–82. Springer Science & Business Media. 50

Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. Computers & Operations Research, 22(1):5–13. 51

Reggelin, T. and Tolujew, J. (2011). A mesoscopic approach to modeling and simulation of logistics processes. In 2011 Winter Simulation Conference - (WSC 2011), pages 1508–1518. 31, 32

Reisman, A., Kumar, A., and Motwani, J. (1997). Flowshop scheduling/sequencing research: a statistical review of the literature, 1952-1994. IEEE Transactions on Engineering Management, 44(3):316–329. 3, 4, 77, 79, 80, 81

Remesh, A. (2021). A hybrid job scheduling approach on cloud computing environments: on the usage of heuristics and metaheuristics methods. Msc thesis, Faculty of Computer Science, Otto-von-Guericke-Universität Magdeburg, Germany, Magdeburg. Assessed and supervised by Turowski, K., Schallehn, E., and Nahhas, A. 170

Remesh, A., Nahhas, A., Kharitonov, A., and Turowski, K. (2022). Investigating different optimization criteria for a hybrid job scheduling approach based on heuristics and metaheuristics. In Australasian Conference on Information Systems (ACIS). 14, 21, 29, 152, 170, 171

Remesh, A., Nahhas, A., Kharitonov, A., and Turowski, K. (2023). A hybrid job scheduling approach on cloud computing environments on the usage of heuristics and metaheuristics methods. In Bui, T. X. (ed.), Proceedings of the 56th Annual Hawaii International Conference on System Sciences, Honolulu, HI. 3, 14, 21, 152, 170, 171

Reza Hejazi, S. and Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: A review. International Journal of Production Research, 43(14):2895–2929. 33, 35

Ribas, I., Leisten, R., and Framiñan, J. M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Computers & Operations Research, 37(8):1439–1454. 2, 29, 33, 35, 81, 82

Rolf, B., Reggelin, T., Nahhas, A., Lang, S., and Müller, M. (2020a). Assigning dispatching rules using a genetic algorithm to solve a hybrid flow shop scheduling problem. Procedia Manufacturing, 42:442–449. 2, 3, 16

Rolf, B., Reggelin, T., Nahhas, A., Muller, M., and Lang, S. (2020b). Scheduling jobs in a two-stage hybrid flow shop with a simulation-based genetic algorithm and standard dispatching rules. In Bae, K.-H. G., Feng, B., Kim, S., Lazarova-Molnar, S., Zheng, Z., Roeder, T., and Thiesing, R. (eds.), Proceedings of the 2020 Winter Simulation Conference (WSC'20), Piscataway, NJ, USA, pages 1584–1595. 2, 17

Romero-Silva, R., Santos, J., and Hurtado-Hernández, M. (2022). A conceptual framework of the applicability of production scheduling from a contingency theory approach: addressing the theory-practice gap. Production Planning & Control, pages 1–21. 3, 4, 45, 77, 79, 80, 81, 84

Ronald, S. (1997). Robust encodings in genetic algorithms: a survey of encoding issues. In Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97), Piscataway N.J., pages 43–48. 123, 124

Ross, P. (2005). Hyper-heuristics. In Burke, E. K. and Kendall, G. (eds.), Search Methodologies, pages 529–556. Springer US, Boston, MA. 2, 3, 4, 5, 45, 48, 79, 82, 84, 86

Ruiz, R. and Maroto, C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research, 169(3):781–800. 50, 51

Ruiz, R. and Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. European Journal of Operational Research, 205(1):1–18. 2, 4, 35, 50, 77, 81, 82

Sartoretti, G., Paivine, W., Shi, Y., Wu, Y., and Choset, H. (2019). Distributed learning of decentralized control policies for articulated mobile robots. IEEE Transactions on Robotics, 35(5):1109–1122. 61

Schrage, L. (1968). Letter to the editor—a proof of the optimality of the shortest remaining processing time discipline. Operations Research, 16(3):687–690. 36

Schrage, L. E. and Miller, L. W. (1966). The queue m / g /1 with the shortest remaining processing time discipline. Operations Research, 14(4):670–684. 36

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2017a). Trust region policy optimization. 58, 59

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms. URL https://arxiv.org/pdf/1707.06347. 59, 60, 136

Seidl, M., Scholz, M., Huemer, C., and Kappel, G. (2015). Uml @ classroom: An introduction to object-oriented modeling. Springer International Publishing and Imprint: Springer, Cham. 91

Selvarajah, E. and Zhang, R. (2014). Supply chain scheduling at the manufacturer to minimize inventory holding and delivery costs. International Journal of Production Economics, 147:117–124. 29

Senthil Kumar, K. and Anandamurugan, S. (2023). An energy and deadline-aware scheduler with hybrid optimization in virtualized clouds. Journal of Electrical Engineering & Technology, 18(6):4415–4424. 81

şerifoğlu, F. S. and Ulusoy, G. (2004). Multiprocessor task scheduling in multistage hybrid flow-shops: a genetic algorithm approach. Journal of the Operational Research Society, 55(5):504–512. 51, 52

Sharvari, T. and SowmyaNag, K. (2019). A study on modern messaging systems- kafka, rabbitmq and nats streaming. ArXiv, abs/1912.03715. 158

Shin, J., Arroyo, D., Tantawi, A., Wang, C., Youssef, A., and Nagi, R. (2022). Cloud-native workflow scheduling using a hybrid priority rule and dynamic task parallelism. In Gavrilovska, A. (ed.), Proceedings of the 13th Symposium on Cloud Computing, New York,NY,United States, pages 72–77. 41

Siboo, S., Bhattacharyya, A., Naveen Raj, R., and Ashwin, S. H. (2023). An empirical study of ddpg and ppo-based reinforcement learning algorithms for autonomous driving. IEEE Access, 11:125094–125108. 60

Silva Filho, M. C., Oliveira, R. L., Monteiro, C. C., Inácio, P. R. M., and Freire, M. M. (2017). Cloudsim plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. In 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pages 400–406. 44, 45, 151

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016a). Mastering the game of go with deep neural networks and tree search. Nature, 529:484–489. 57, 62

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016b). Mastering the game of go with deep neural networks and tree search. Nature, 529(7587):484–489. 64

Silver, D., Lever, G., Heess, N. M. O., Degris, T., Wierstra, D., and Riedmiller, M. A. (2014). Deterministic policy gradient algorithms. In International Conference on Machine Learning. 59

Sindhu, S. and Mukherjee, S. (2011). Efficient task scheduling algorithms for cloud computing environment. In Mantri, A. (ed.), High Performance Architecture and Grid Computing, Heidelberg and New York, pages 79–83. 40

Sipser, M. (2012). Introduction to the theory of computation. Cengage Learning. 32

Smed, J., Johnsson, M., Johtela, T., and Nevalainen, O. (2003). Techniques and applications of production planning in electronics manufacturing systems. In Leondes, C. T. (ed.), Computer aided and integrated manufacturing systems:, pages 1–48. World Scientific, New Jersey and London. 171

Song, W., Chen, X., Li, Q., and Cao, Z. (2023). Flexible job-shop scheduling via graph neural network and deep reinforcement learning. IEEE Transactions on Industrial Informatics, 19(2):1600–1610. 84

Sonnenberg, C. and vom Brocke, J. (2012). Evaluations in the science of the artificial – reconsidering the build-evaluate pattern in design science research. In Peffers, K., Rothenberger, M., and Kuechler, W. (eds.), Design science research in information systems, pages 381–397. Springer, Berlin. 8, 18, 139, 140, 143, 162, 171, 199, 206

Sousa, P. S. A. and Moreira, M. R. A. (2007). Performance analysis of job-shop production systems under different order release control parameters. In Ao, S.-I. (ed.), Lecture Notes in Electrical Engineering, pages 1056–1061. IAENG, London. 23

Spanos, A. C., Gayialis, S. P., Kechagias, E. P., and Papadopoulos, G. A. (2022). An application of a decision support system enabled by a hybrid algorithmic framework for production scheduling in an sme manufacturer. Algorithms, 15(10):372. 39

Srinivas, N. and Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evol. Comput., 2(3):221–248. 53, 54

Stanley, K. O. and Miikkulainen, R. (2002). Efficient evolution of neural network topologies. In Congress on evolutionary computation, pages 1757–1762. 197

Stansfield, W. D. (1991). Schaum's outline of theory and problems of genetics. McGraw-Hill, New York. 123

Stützle, T. (1998). An ant approach to the flow shop problem. In Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing, pages 1560–1564. 33, 35

Sun, Y. and Kantor, P. B. (2006). Cross-evaluation: A new model for information system evaluation. Journal of the American Society for Information Science and Technology, 57(5):614–628. 140

Sun, Z., Li, Z., Gu, C., and Huang, H. (2024). An energy-efficient scheduling method

for real-time multi-workflow in container cloud. In Wu, W. and Guo, J. (eds.), COMBINATORIAL OPTIMIZATION AND APPLICATIONS, pages 168–181. SPRINGER INTERNATIONAL PU, [S.l.]. 45, 163

Sutton, R. S. and Barto, A. (2018a). Reinforcement learning: An introduction. The MIT Press, Cambridge, Massachusetts and London, England. 59

Sutton, R. S. and Barto, A. G. (2018b). Reinforcement learning: An introduction. The MIT Press. 56, 58

Swan, J., Adriaensen, S., Brownlee, A. E., Hammond, K., Johnson, C. G., Kheiri, A., Krawiec, F., Merelo, J. J., Minku, L. L., Özcan, E., Pappa, G. L., García-Sánchez, P., Sörensen, K., Voß, S., Wagner, M., and White, D. R. (2022). Metaheuristics "in the large". European Journal of Operational Research, 297(2):393–406. 4, 50, 77, 82, 83

Szepesvari, C. (2010). Algorithms for reinforcement learning. In Synthesis Lectures on Artificial Intelligence and Machine Learning. 56

Szepesvári, C. (2022). Algorithms for reinforcement learning. Springer, Cham. 58

Szita, I. and Lörincz, A. (2006). Learning tetris using the noisy cross-entropy method. Neural Comput., 18(12):2936–2941. 58

Szita, I. and Lörincz, A. (2006). Learning tetris using the noisy cross-entropy method. Neural Computation, 18(12):2936–2941. 58

Talbi, E.-G. (2009). Metaheuristics: From design to implementation. John Wiley & Sons, Hoboken, NJ. 123, 124, 129

Tesauro, G. (1995). Temporal difference learning and td-gammon. Commun. ACM, 38(3):58–68. 57

Tizhoosh, H. R. (2006). Opposition-based learning: A new scheme for machine intelligence. In Mohammadian, M. (ed.), CIMCA 2005 jointly with IAWTIC 2005, Los Alamitos Calif., pages 695–701. 69

Turowski, K. (2001). Spezifikation und standardisierung von fachkomponenten. Wirtschaftsinformatik, 43(3):269–281. 8, 10

Turowski, K. (2003). Fachkomponenten: Komponentenbasierte betriebliche anwendungssysteme. Shaker, Aachen. 8, 10, 88

Urquhart, N., Guckert, M., and Powers, S. (2019). Increasing trust in meta-heuristics by using map-elites. In López-Ibáñez, M., Auger, A., and Stützle, T. (eds.), GECCO' 19, New York, pages 1345–1348. 4, 77, 84

Varasteh, A. and Goudarzi, M. (2017). Server consolidation techniques in virtualized data centers: A survey. IEEE Systems Journal, 11(2):772–783. 30, 81, 136

Venable, J. (2006). A framework for design science research activities. Emerging Trends and Challenges in Information Technology Management: Proceedings of the 2006 Information Resource Management Association Conference, pages 184–187. 140

Vesikar, Y., Deb, K., and Blank, J. (2018). Reference point based nsga-iii for preferred solutions. In 2018 IEEE Symposium Series on Computational Intelligence (SSCI), pages 1587–1594. 54

vom Brocke, J., Simons, A., Niehaves, B., Reimer, K., Plattfaut, R., and Cleven, A. (2009). Reconstructing the giant: On the importance of rigour in documenting the literature search process. ECIS 2009 Proceedings. 161. 63

von Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. MIS quarterly, 28(1):75–105. 6, 7, 8, 9, 18, 79, 85, 87, 139, 162, 199

Voß, S. (1993). The two — stage hybrid — flowshop scheduling problem with sequence — dependent setup times. In Fandel, G., Gulledge, T., and Jones, A. (eds.), Operations Research in Production Planning and Control, pages 336–352. Springer Berlin Heidelberg. 47

Voudouris, C. and Tsang, E. (2003). Guided local search. In Glover, F. and Kochenberger, G. A. (eds.), Handbook of Metaheuristics, pages 185–218. Springer Science & Business Media. 48

Wang, Q., Xiong, J., Han, L., sun, p., Liu, H., and Zhang, T. (2018). Exponentially weighted imitation learning for batched historical data. Advances in Neural Information Processing Systems, 31. 15, 211

Wang, S., Li, J., Tang, H., and Wang, J. (2022a). Cea-fjsp: Carbon emission-aware flexible job-shop scheduling based on deep reinforcement learning. Frontiers in Environmental Science, 10. 74, 75

Wang, X., Wang, S., Liang, X., Zhao, D., Huang, J., Xu, X., Dai, B., and Miao, Q. (2022b). Deep reinforcement learning: A survey. IEEE Transactions on Neural Networks and Learning Systems, pages 1–15. 59

Wang, Y., Jia, Z.-h., and Li, K. (2021). A multi-objective co-evolutionary algorithm of scheduling on parallel non-identical batch machines. Expert Systems with Applications, 167:114145. 177

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2016).

Dueling network architectures for deep reinforcement learning. 57

Waterman, A. S. (2016). Design of the risc-v instruction set architecture. University of California, Berkeley. 144

Waubert de Puiseau, C., Meyes, R., and Meisen, T. (2022). On reliability of reinforcement learning based production scheduling systems: a comparative survey. Journal of Intelligent Manufacturing, 33:1–17. 57

Webster, J. and Watson, R. T. (2002). Analyzing the past to prepare for the future: Writing a literature review. MIS Quarterly, Vol. 26, No. 2 (Jun. 2002), pages 13–23. 63

Weng, W., Chen, J., Zheng, M., and Fujimura, S. (2022). Realtime scheduling heuristics for just-in-time production in large-scale flexible job shops. Journal of Manufacturing Systems, 63:64–77. 39

Whitley, D. (1994). A genetic algorithm tutorial. Statistics and Computing, 4(2):65–85. 50, 51

Wierstra, D., Schaul, T., Peters, J., and Schmidhuber, J. (2008). Natural evolution strategies. In 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), pages 3381–3387. 58

Williams, R. J. (2004). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8:229–256. 58

Witanto, J. N., Lim, H., and Atiquzzaman, M. (2018). Adaptive selection of dynamic vm consolidation algorithm using neural network for cloud resource management. Future Generation Computer Systems, 87:35–42. 169

Wittrock, R. J. (1990). Scheduling parallel machines with major and minor setup times. International Journal of Flexible Manufacturing Systems, 2(4):329–341. 24, 30, 45, 46, 49

Wongchai, A., Parvati, V. K., Al-Safarini, M. Y., Shamsi, W. D., Singh, B., and Huy, P. Q. (2022). Manufacturing industry-based optimal scheduling method of information system operation and maintenance resources. International Journal of Advanced Manufacturing Technology. 25

Wu, Y., Liu, L., Bae, J., Chow, K.-H., Iyengar, A., Pu, C., Wei, W., Yu, L., and Zhang, Q. (2019). Demystifying learning rate policies for high accuracy training of deep neural networks. 193

Xu, J. and Fortes, J. A. B. (2010). Multi-objective virtual machine placement in virtualized data center environments. In 2010 IEEE/ACM Int'l Conference on Green Computing

and Communications & Int'l Conference on Cyber, Physical and Social Computing, pages 179–188. 52

Yan, H.-S. and Li, W.-C. (2017). A multi-objective scheduling algorithm with self-evolutionary feature for job-shop-like knowledgeable manufacturing cell. Journal of Intelligent Manufacturing, 28(2):337–351. 71, 75

Yang, X.-S. (2021). Nature-inspired optimization algorithms. Academic Press, London and San Diego, CA. 84

Ye, X., Li, J., Liu, S., Liang, J., and Jin, Y. (2019). A hybrid instance-intensive workflow scheduling method in private cloud environment. Natural Computing, 18(4):735–746. 52

Yi, J. and Liu, X. (2023). Deep reinforcement learning for intelligent penetration testing path design. Applied Sciences, 13(16). 57

Yin, L., Zhuang, M., Jia, J., and Wang, H. (2020). Energy saving in flow-shop scheduling management: An improved multiobjective model based on grey wolf optimization algorithm. Mathematical Problems in Engineering, 2020. 72, 75

Yue, M. (1991). A simple proof of the inequality ffd (l) $\leq$ 11/9 opt (l) + 1, $\forall$ l for the ffd bin-packing algorithm. Acta Mathematicae Applicatae Sinica, 7(4):321–331. 42, 44

Zeiträg, Y., Rui Figueira, J., and Figueira, G. (2024). A cooperative coevolutionary hyper-heuristic approach to solve lot-sizing and job shop scheduling problems using genetic programming. International Journal of Production Research, pages 1–28. 177

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2021). Understanding deep learning (still) requires rethinking generalization. Communications of the ACM, 64(3):107–115. 15

Zhang, C., Vinyals, O., Munos, R., and Bengio, S. (2018). A study on overfitting in deep reinforcement learning. URL https://arxiv.org/pdf/1804.06893. 15, 197, 211

Zhang, W. and Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, San Francisco, CA, USA, page 1114–1120. 57

Zheng, D.-Z. and Wang, L. (2003). An effective hybrid heuristic for flow shop scheduling. The International Journal of Advanced Manufacturing Technology, 21(1):38–44. 51

Zhou, T., Tang, D., Zhu, H., and Wang, L. (2021). Reinforcement learning with composite rewards for production scheduling in a smart factory. IEEE Access, 9:752–766. 72, 75

Zhou, T., Zhu, H., Tang, D., Liu, C., Cai, Q., Shi, W., and Gui, Y. (2022). Reinforcement

learning for online optimization of job-shop scheduling in a smart manufacturing factory. Advances in Mechanical Engineering, 14(3). 73, 75

Đurasević, M. and Jakobović, D. (2020). Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment. Applied Soft Computing, 96:106637. 38