POLITECNICO DI TORINO

School of Management and Production Engineering

Master of Science Course in
Engineering and Management

Master of Science Thesis

A reinforcement learning algorithm for Dynamic Job Shop Scheduling

Supervisor
Prof. Giulia Bruno

Co – supervisor
Niccolò Giovenali

Candidate
Laura Alcamo

Academic year 2023/2024
Graduate session July 2024

Acknowledgements

Completing this thesis has been both challenging and rewarding, and I am deeply grateful to everyone who supported me throughout this journey.

I would like to express my deepest gratitude to my supervisor, Giulia Bruno, whose guidance, support, and encouragement were invaluable throughout the entire process of researching and writing. Her expertise, patience, and insightful feedback have significantly shaped this work and my academic journey.
I am also thankful to my co-supervisor, Niccolò Giovenali, for his contributions, constructive criticism, and scholarly guidance that enriched the depth and quality of this thesis.
I extend my thanks to all those who contributed to this thesis, whether through discussions, feedback, or simply being there when needed. Your support has been indispensable and I am sincerely grateful for your contributions.

Finally, I am deeply grateful to my family and friends for their support, understanding and patience throughout this academic pursuit. Their support has been a source of strength and motivation.

Thank you all for being part of this enriching academic journey.

Laura Alcamo

Abstract

In the era of Industry 4.0, optimizing production processes has become increasingly critical due to the high demand for efficiency, flexibility, and customization in manufacturing.
The Job Shop Scheduling Problem (JSSP), a prominent NP-hard problem, plays a pivotal role in this context, requiring the scheduling of jobs with multiple operations on specific machines in a predetermined order. Effective solutions to JSSP are essential for minimizing production time, reducing costs, and enhancing overall productivity.

This thesis presents the development and evaluation of a single-agent reinforcement learning algorithm designed to address both the JSSP and its dynamic variant (DJSSP).
The primary objective of this research is to test the efficiency and adaptability of reinforcement learning algorithm for scheduling solutions in both deterministic and dynamic environments characterized by variability and uncertainty.

The proposed reinforcement learning approach autonomously learns optimal scheduling policies through iterative interactions with the scheduling environment, dynamically adapting to changes and unexpected disruptions. The algorithm's performance is rigorously benchmarked against traditional scheduling methods, including First-Come, First-Served (FCFS), Shortest Processing Time (SPT), and Genetic Algorithms (GA).

Empirical results demonstrate that the reinforcement learning algorithm is comparable to traditional scheduling methods in the deterministic case, while outperforms conventional techniques in dynamic environments, exhibiting superior adaptability and efficiency across various scheduling scenarios.

These findings underscore the significant potential of AI-driven methodologies to improve operational efficiency in complex scheduling tasks, offering valuable contributions to manufacturing, logistics, and other industries where optimal resource allocation is paramount.

Table of Contents

List of figures

List of tables

Introduction

In recent years, the advent of Industry 4.0 and Industry 5.0 has markedly transformed the manufacturing and production sectors. Industry 4.0 is distinguished by the integration of cyber-physical systems, Internet of Things (IoT), and big data analytics, with the objective of establishing smart factories. In these environments, machines and systems are interconnected and capable of autonomous communication to optimize production processes.

A pivotal enabler within the frameworks of Industry 4.0 and Industry 5.0 is the deployment of advanced machine learning algorithms. These algorithms hold substantial potential across various industries by learning from data and subsequently making predictions or classifications based on acquired knowledge. Industry 5.0 advances this paradigm by focusing on human-machine collaboration, sustainable manufacturing practices, and the customization of products.
The transition from traditional manufacturing models to smart factories under Industry 4.0 has presented a range of challenges and opportunities, fostering significant advancements in production efficiency and innovation.

One of the critical challenges is the high demand for efficiency, flexibility, and customization in manufacturing.
The job shop scheduling problem, a notable NP-hard problem, requires scheduling jobs with multiple operations on specific machines in a predetermined order. A strong assumption is that all the information of the manufacturing environment is known in advance and there is no modification during the scheduling process. However, the real-world environment is significantly affected by uncertainties. The dynamic job shop scheduling is a variant of the job shop scheduling problem in which the scheduling environment is subject to changes over time including variations in job arrival times, processing times, machine break- downs, resource availability and job priority.

Traditional approaches to solving JSSP, such as heuristic and enumerative methods, often fall short in dynamic settings due to their inability to adapt to changes effectively.
Reinforcement learning (RL), a subfield of artificial intelligence, offers a promising alternative by enabling systems to learn and adapt to dynamic environments. RL algorithms train agents to make decisions through trial-and-error interactions with the environment, optimizing a cumulative reward signal over time. This adaptive capability makes RL particularly suitable for addressing the complexities of DJSSP.

Objectives

The primary focus of this thesis is the development of a reinforcement learning algorithm to address the Dynamic Job Shop Scheduling Problem (DJSSP). The goal is to design a system that can adapt to changes in the job shop environment, such as new job arrivals, and variable processing times, while optimizing the overall performance of the scheduling process.

The objectives of this thesis are as follows:

- Develop a Single-Agent RL Scheduler: Design and implement a single-agent reinforcement learning algorithm tailored to the DJSSP, leveraging state-of-the-art DRL techniques for improved stability and performance.
- Incorporate Dynamic Elements: Integrate dynamic elements into the scheduling environment to create a realistic and challenging DJSSP scenario.
- Optimize Scheduling Performance: Evaluate the proposed RL algorithm against established benchmarks, focusing on key performance metrics.
- Compare with Traditional Methods: Conduct a comparative analysis with traditional heuristic and exact methods to highlight the advantages and limitations of the RL approach in dynamic settings.

The proposed methodology involves several key steps:

1. Problem Formulation: Define the DJSSP as a Markov Decision Process (MDP), specifying the state space, action space, and reward function.
2. Algorithm Development: Develop a RL algorithm to tackle the Job Shop Scheduling Problem and its dynamic variant
3. Simulation and Training: Create a simulated job shop environment to train the RL agent, using realistic job shop scenarios to enhance the training process.
4. Evaluation and Analysis: Evaluate the trained RL agent on a set of benchmarks DJSSP instances, comparing its performance with traditional methods and analyzing its adaptability to dynamic changes.

This thesis aims to contribute to the field of job shop scheduling and reinforcement learning in several ways:

- Novel RL Framework: Introduce a novel single-agent RL framework for DJSSP, demonstrating its effectiveness and adaptability in dynamic scheduling environments.
- Enhanced Performance: Show that the proposed RL algorithm can outperform traditional heuristic and exact methods in terms of key performance metrics under dynamic conditions.
- Scalability and Practicality: Provide insights into the scalability and practicality of RL approaches for real-world manufacturing and production environments, highlighting potential applications and future research directions.

Structure of the thesis

This thesis is organized into ten chapters that systematically explore the integration of Reinforcement Learning in the context of the Job Shop Scheduling Problem within the Industry 4.0 paradigm.

The first chapter provides a comprehensive overview of Industry 4.0, tracing the evolution from earlier industrial revolutions and defining the key features, impacts, and challenges of Industry 4.0.
The second chapter delves into scheduling, highlighting its significant economic, environmental, and customer satisfaction impacts, and discussing the challenges associated with scheduling and underscores its growing importance in the context of Industry 4.0.
Chapter three focuses on the Job Shop Scheduling Problem itself, offering a detailed problem overview including the historical context, and the relevance of JSSP in contemporary manufacturing.
Reinforcement Learning is introduced in chapter four, covering the basics of artificial intelligence and machine learning, and providing an in-depth overview of RL algorithms. Building on this foundation, chapter five explores Deep Reinforcement Learning (DRL), introducing deep learning concepts and specific DRL algorithms such as Deep Q-learning and Proximal Policy Optimization.
The literature review in chapter seven synthesizes existing research, setting the context for the novel contributions of this thesis. Chapter eight details the development of a specific RL algorithm for JSSP, discussing both theoretical and practical implementation.
Chapters nine and ten present empirical results of the RL algorithm application to both deterministic and dynamic environments.
The thesis concludes with an analysis of the findings of the research, and an outline of potential directions for future work in this domain.

1 Industry 4.0 overview

1.1 Industrial revolutions, from 1.0 to 4.0

The term "Industry 4.0", coined by the German government in 2011 refers to the fourth industrial revolution, characterized by the integration of advanced digital technologies into manufacturing and production processes [1]. This concept encompasses cutting-edge technologies aiming to create interconnected and intelligent production environments [2]. The evolution of Industry 4.0 marks a significant transformation from traditional manufacturing to smart factories, where machines and systems communicate to enhance efficiency and productivity [3].

The journey to Industry 4.0 is rooted in a series of historical industrial revolutions that have profoundly shaped manufacturing and production processes over time (Figure 1).

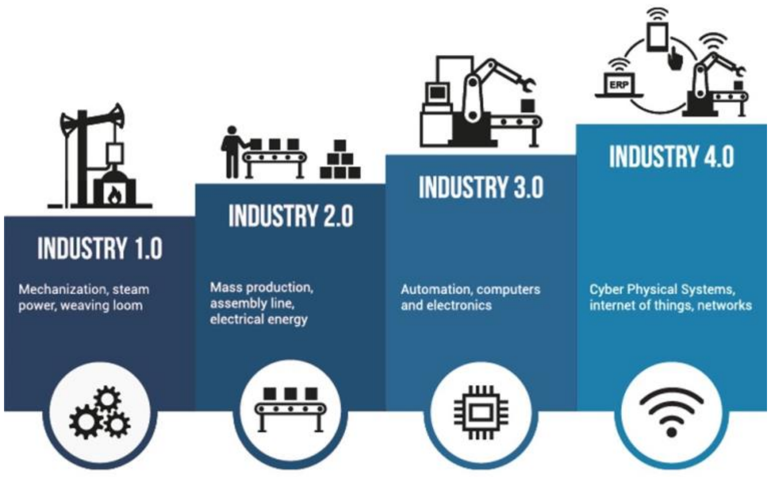

INDUSTRY 1.0
Mechanization, steam power, weaving loom
INDUSTRY 2.0
Mass production, assembly line, electrical energy
INDUSTRY 3.0
Automation, computers and electronics
INDUSTRY 4.0
Cyber Physical Systems, internet of things, networks
ERP

Figure 1: Four industrial revolutions [1]

The first industrial revolution, which began in the late 18th century, introduced mechanization through the utilization of water and steam power. This transformative shift from manual labor to machine-based production was pivotal in enabling mass production and setting the stage for industrialization.

Building upon these foundations, the second industrial revolution emerged in the late 19th and early 20th centuries with the widespread adoption of electricity and the development of assembly line techniques. Electricity replaced steam as the primary source of power in factories, significantly enhancing productivity and operational flexibility. The introduction of assembly lines by pioneers such as Henry Ford revolutionized manufacturing by streamlining production processes and enabling efficient mass production on a large scale.

The third industrial revolution, beginning in the mid-20th century, marked the advent of automation and computerization in manufacturing. This era saw the integration of computers into industrial processes, facilitating greater precision, control, and automation of tasks.

Additionally, advancements in telecommunications and the rise of the Internet during this period revolutionized global connectivity and digital information exchange, laying the groundwork for further technological advancements.[4]

In the early 21st century, the fourth industrial revolution, known as Industry 4.0, emerged with the convergence of digital technologies and advanced analytics. In this phase, characterized by the digitalization of manufacturing and the development of smart factories, machines, systems, and humans communicate and collaborate seamlessly, leading to autonomous decision-making and real-time optimization of production processes.

Each industrial revolution has built upon the achievements and innovations of its predecessors, driving continuous advancements in manufacturing capabilities and reshaping economic and societal landscapes. Industry 4.0 represents a transformative shift towards interconnected and intelligent production systems, poised to further accelerate innovation, efficiency, and economic growth in the digital age.

Defining Industry 4.0

Industry 4.0 signifies a transformative shift in manufacturing, characterized by the convergence of physical and digital technologies that enable smarter, more efficient production processes.

- Cyber-Physical Systems (CPS).

  CPS integrate physical processes with computational algorithms and real-time data analytics [5]. These systems monitor and control physical operations, such as machinery and production lines, enabling autonomous decision-making and optimization of manufacturing processes. CPS are essential for achieving higher levels of automation and responsiveness in smart factories.

- Internet of Things (IoT).

  IoT networks connect devices, sensors, and equipment within manufacturing environments, enabling seamless data exchange and remote monitoring [6]. In Industry 4.0, IoT facilitates continuous data collection from various points in the production chain, enabling predictive maintenance, real-time quality monitoring, and resource optimization.

- Big Data and Analytics.

  The proliferation of CPS and IoT devices generates vast amounts of data that are analyzed using advanced analytics and AI techniques [1]. Big data analytics in Industry 4.0 enable manufacturers to extract actionable insights, optimize production workflows, and enhance decision-making processes based on real-time data trends and patterns [7].

- Artificial Intelligence (AI) and Machine Learning (ML).

AI and ML algorithms are integral to Industry 4.0 for automating complex tasks, predicting outcomes, and optimizing operations [8]. AI-driven systems can adapt and learn from data, improving efficiency, quality control, and predictive maintenance capabilities across manufacturing processes.

- Cloud Computing.

  Cloud platforms provide scalable infrastructure for storing, processing, and analyzing large volumes of data generated by Industry 4.0 technologies [9]. Cloud computing facilitates real-time collaboration, remote monitoring, and access to computational resources, enabling manufacturers to deploy and manage applications efficiently.

- Additive Manufacturing (3D Printing).

  Additive manufacturing technologies enable the production of customized, intricate parts with reduced material waste and lead time [10]. 3D printing in Industry 4.0 enhances flexibility in manufacturing, allowing for rapid prototyping, on-demand production, and complex geometries that traditional manufacturing methods cannot achieve.

- Augmented Reality (AR) and Virtual Reality (VR).

  AR and VR technologies enhance training, maintenance, and design processes in manufacturing [11]. These immersive technologies provide real-time visualizations, virtual simulations, and interactive guidance that improve operational efficiency, training effectiveness, and safety protocols within smart factories.

1.2 Impact and Benefits of Industry 4.0

The implementation of Industry 4.0 technologies in manufacturing brings about several transformative benefits that enhance efficiency, flexibility, quality, and decision-making processes across industries.

- Increased Efficiency and Productivity: Industry 4.0 integrates real-time data collection and analysis through advanced sensors and IoT devices [12]. These technologies enable manufacturers to monitor operations continuously, optimize production schedules, and minimize downtime by identifying and resolving inefficiencies promptly [13]. By leveraging data-driven insights, companies can improve resource allocation, streamline workflows, and ultimately enhance overall productivity [7].

- Enhanced Flexibility and Customization: Smart factories equipped with cyber-physical systems can dynamically adjust production processes in response to changing market demands and customer preferences [13]. Through interconnected machinery and adaptive manufacturing processes, companies can reduce lead times and offer customized products at scale. This capability not only improves customer satisfaction but also strengthens competitive advantage in a rapidly evolving market landscape [14].

- Improved Quality and Reduced Waste: Advanced monitoring and control systems in Industry 4.0 facilitate real-time quality assurance and defect detection [15]. By analyzing data from sensors embedded throughout the production line, manufacturers can detect deviations from quality standards early, leading to immediate corrective actions [15]. This proactive approach not only ensures consistent product quality but also minimizes material waste, contributing to more sustainable manufacturing practices [16].

- Predictive Maintenance: IoT-enabled sensors and AI algorithms enable predictive maintenance strategies by continuously monitoring equipment performance and detecting anomalies [17][18]. Predictive maintenance anticipates equipment failures before they occur, allowing for timely repairs and preventing costly unplanned downtime [7]. By optimizing maintenance schedules based on real-time data insights, manufacturers can extend asset lifespan and reduce overall maintenance costs.

- Better Decision-Making: Industry 4.0 provides decision-makers at all levels of the organization with access to comprehensive, real-time data analytics [5]. Advanced analytics tools, powered by big data and AI, enable accurate forecasting, risk assessment, and performance monitoring [19]. This data-driven approach empowers managers to make informed decisions swiftly, optimize supply chain management, and capitalize on emerging market opportunities [20].

In summary, Industry 4.0 represents a paradigm shift towards interconnected, intelligent manufacturing systems that leverage digital technologies to optimize operations, enhance product quality, and drive innovation. By embracing these advancements, manufacturers can achieve substantial improvements in efficiency, flexibility, and decision-making capabilities, positioning themselves for sustainable growth in the global economy.

1.3 Challenges and Considerations

Despite the transformative potential of Industry 4.0, its adoption poses several significant challenges that organizations must navigate:

- Cybersecurity: The increased connectivity of cyber-physical systems and IoT devices in Industry 4.0 environments heightens the risk of cyberattacks and data breaches. Protecting sensitive data, intellectual property, and operational continuity from cyber threats is a paramount concern. Robust cybersecurity measures, including encryption, authentication protocols, and continuous monitoring, are essential to mitigate these risks and ensure the resilience of digital manufacturing ecosystems [2].

- Integration with Legacy Systems: Many traditional manufacturing systems were not designed with interoperability in mind, posing challenges for integrating new Industry 4.0 technologies [21]. Legacy systems may lack the necessary connectivity or data standards required for seamless integration with modern digital platforms. Overcoming these integration barriers often requires significant investments in retrofitting existing infrastructure or adopting hybrid approaches that bridge legacy and digital technologies [22].

- Skills Gap: The implementation of Industry 4.0 demands a workforce proficient in advanced digital technologies, such as data analytics, AI, and robotics [7]. However, there is a notable shortage of skilled professionals with expertise in these areas. Addressing the skills gap through targeted education, vocational training programs, and upskilling initiatives is crucial to empower workers and enable organizations to fully leverage the capabilities of Industry 4.0[24].

- High Initial Investment: Transitioning to smart manufacturing involves substantial upfront investments in technology infrastructure, equipment upgrades, and workforce training [25]. For small and medium-sized enterprises (SMEs) in particular, the financial burden of adopting Industry 4.0 technologies can be prohibitive. Access to funding, incentives for technological adoption, and collaborative partnerships can help SMEs overcome financial barriers and facilitate their entry into the digital manufacturing landscape [13].

- Data Privacy: The proliferation of IoT devices and the extensive collection of data in Industry 4.0 environments raise concerns about data privacy and ethical use. Organizations must implement robust data governance frameworks to ensure compliance with privacy regulations and protect the confidentiality of sensitive information. Transparent data management practices, informed consent mechanisms, and ethical guidelines for data usage are essential to build trust among stakeholders and mitigate privacy risks associated with digital transformation initiatives [26].

In conclusion, while Industry 4.0 promises significant advancements in efficiency, customization, and decision-making capabilities, addressing cybersecurity risks, legacy system integration, skills shortages, financial barriers, and data privacy concerns is essential to successfully harness its full potential.

1.4 Future Prospects and Industry 5.0

As Industry 4.0 continues to advance, the concept of Industry 5.0 is emerging as the next evolution in manufacturing. Industry 5.0 shifts the focus from purely automated processes to a collaborative ecosystem where humans and machines work together synergistically. This paradigm aims to harness the efficiency and precision of automation while integrating human creativity and innovation into production processes [27]

Industry 5.0 envisions a future where machines handle routine tasks and repetitive processes, allowing human workers to focus on complex problem-solving, creativity, and decision-making. By leveraging advanced technologies such as AI, robotics, and augmented reality, Industry 5.0 aims to enhance the capabilities of human workers, enabling them to interact more intuitively with machines and systems [28].

The human-centric approach of Industry 5.0 underscores the importance of human skills and ingenuity in driving innovation and sustainable manufacturing practices. Unlike previous industrial revolutions that often led to concerns about job displacement, Industry 5.0 emphasizes the augmentation rather than replacement of human labor. This approach ensures that technology serves as a tool to amplify human potential, enabling more personalized and adaptive production processes that can respond dynamically to customer needs and market changes.

By fostering a collaborative environment between humans and machines, Industry 5.0 aims to achieve higher levels of productivity, efficiency, and quality in manufacturing. This synergy allows for greater flexibility and customization in production, leading to improved customer satisfaction and sustainability outcomes.

2 Scheduling

Scheduling is a fundamental aspect of operations management, playing a crucial role in the efficient allocation of resources, the optimization of processes, and the timely completion of tasks. As organizations and systems grow in complexity, effective scheduling becomes increasingly vital to maintain productivity and competitiveness. This thesis explores the intricacies of scheduling, examining various methodologies, tools, and applications that contribute to enhanced operational performance.
The concept of scheduling encompasses a broad spectrum of activities, from simple task allocation to intricate project management in diverse fields such as manufacturing, healthcare, transportation, and information technology. Effective scheduling ensures that resources, including time, labor, and equipment, are utilized optimally, minimizing downtime and maximizing output. The ability to prioritize tasks, allocate resources judiciously, and adapt to unforeseen changes is central to achieving operational excellence. [29]

In manufacturing, efficient job scheduling is essential for maintaining smooth production flows and meeting customer demands. Manufacturers strive to sequence production tasks in a way that minimizes setup times, reduces idle machinery, and maximizes throughput.

In logistics, effective scheduling ensures timely delivery of goods and services while minimizing transportation costs and optimizing route efficiency. Logistics companies utilize scheduling algorithms to coordinate vehicle dispatching, route planning, and inventory management. Real-time scheduling systems enable adjustments based on traffic conditions, weather forecasts, and customer priorities, enhancing overall logistics performance and customer satisfaction.

In healthcare, efficient job scheduling is critical for managing patient appointments, operating room schedules, and healthcare staff assignments. Healthcare providers use scheduling algorithms to optimize the allocation of resources such as doctors, nurses, and medical equipment, ensuring efficient patient care delivery and reducing wait times. Dynamic scheduling systems adapt to changes in patient demand and medical emergencies, improving healthcare service quality and operational efficiency.

In information technology (IT), job scheduling automates the execution of tasks such as data backups, system updates, and batch processing. IT departments rely on scheduling algorithms to optimize server utilization, prioritize critical tasks, and minimize downtime. Job scheduling software integrates with enterprise systems to streamline workflow management, enhance data security measures, and ensure continuous IT service availability [29].

The complexity of job and task scheduling escalates with the scale of operations and the interdependencies among tasks and resources. To address these challenges, industries employ sophisticated scheduling methods, including mathematical modeling, heuristic algorithms, and artificial intelligence techniques [30]. These approaches enable organizations to devise efficient schedules that balance competing objectives and adapt to dynamic operational environments.
Job and task scheduling is integral to optimizing operational efficiency and achieving strategic goals across manufacturing, logistics, healthcare, and IT sectors. By leveraging advanced

scheduling techniques and algorithms, industries can enhance productivity, reduce costs, and improve service delivery, thereby gaining a competitive edge in today's dynamic business landscape.

2.1 Scheduling impact

2.1.1 Economic

The economic importance of scheduling cannot be overstated in today's fast-paced and competitive business environment. Efficient scheduling practices directly influence a company's financial performance by minimizing idle times, optimizing throughput, and reducing operational costs [31].

The impact of scheduling on economic performance is evident across various industries.
In manufacturing, effective job scheduling plays a pivotal role in ensuring the smooth and continuous flow of production processes. By minimizing downtime and maximizing machine utilization, manufacturers can achieve higher levels of productivity and profitability. For example, proper scheduling aligns production tasks with available resources, reducing setup times and ensuring optimal use of equipment and labor resources [32].

In the service sector, optimized scheduling can improve customer satisfaction and service delivery efficiency, leading to increased revenue. Moreover, in sectors such as healthcare and transportation, effective scheduling is crucial for balancing demand and supply, thereby improving service quality and economic outcomes.

The financial benefits of efficient job scheduling extend beyond immediate cost reductions. They enable companies to reinvest saved resources into innovation, infrastructure improvements, and talent development, thereby enhancing their competitive position in the market [33]. By continuously optimizing scheduling practices, organizations can adapt to market dynamics, improve service quality, and maintain sustainable growth in the long term.

2.1.2 Environmental sustainability

The concept of environmental sustainability has become a cornerstone in contemporary business practices and operational strategies. Effective scheduling, traditionally associated with optimizing resource allocation and enhancing productivity, now also plays a crucial role in promoting environmental sustainability.
Efficient job and task scheduling can yield significant benefits by reducing energy consumption, minimizing waste, and lowering carbon emissions across various industries. [

In energy-intensive manufacturing sectors, scheduling algorithms play a crucial role in optimizing production processes to align with periods of low energy costs or high renewable energy availability [34]. By strategically scheduling operations during off-peak hours or times when renewable energy sources like solar or wind are abundant, industries can reduce their reliance on fossil fuels and lower their carbon footprint. This approach not only reduces operational costs but also contributes to environmental conservation efforts by mitigating greenhouse gas emissions associated with energy consumption (Fang et al., 2011).

Efficient logistics scheduling minimizes the environmental impact of transportation activities. By optimizing delivery routes, vehicle dispatching, and load consolidation, logistics companies can reduce the number of vehicles on the road, thereby lowering fuel consumption and emissions [36]. Advanced scheduling systems enable logistics providers to prioritize eco-friendly transport modes, such as rail or electric vehicles, further reducing the carbon intensity of their operations.

The environmental benefits of effective scheduling extend beyond energy efficiency to encompass waste reduction and resource conservation. By streamlining production schedules and optimizing resource utilization, industries can minimize material waste and enhance resource efficiency [29].

The environmental benefits of effective scheduling extend beyond mere compliance with regulations. They align with the broader agenda of corporate social responsibility (CSR) and the growing consumer demand for sustainable practices. Companies that adopt environmentally sustainable scheduling practices can enhance their brand reputation, attract eco-conscious customers, and achieve long-term economic benefits through cost savings and improved efficiency.

2.1.3 Customer satisfaction

In the modern business landscape, customer satisfaction is a vital component of success, deeply influencing loyalty, reputation, and profitability. Central to achieving high levels of customer satisfaction is the implementation of effective scheduling practices. By ensuring timely and reliable service delivery, optimal resource utilization, and the consistent fulfillment of customer expectations, strategic scheduling plays an essential role in enhancing the overall customer experience [37].

The direct impact of scheduling on customer perceptions and experiences is profound.
For example, in healthcare, reducing patient waiting times and improving patient flow through meticulous scheduling can significantly enhance the patient experience. (Gupta & Denton, 2008). Scheduling algorithms help healthcare facilities allocate medical staff, operating rooms, and equipment efficiently, ensuring that resources are available when needed to meet patient demand. This enhances the quality of care provided, leading to higher patient satisfaction and better health outcomes [38].

Moreover, in the field of information technology (IT), job scheduling algorithms are instrumental in managing computational tasks and optimizing resource utilization in cloud computing environments. Cloud service providers rely on efficient scheduling techniques to allocate computing resources dynamically, ensuring that applications and services run smoothly with minimal downtime. By scheduling tasks effectively, IT organizations can meet service level agreements (SLAs), maintain high system availability, and enhance user satisfaction with reliable and responsive IT services [39].

2.2 Scheduling challenges

Despite its importance, scheduling is fraught with numerous challenges that can significantly impact organizational performance. At its core, scheduling involves the strategic planning and coordination of tasks, resources, and time to achieve optimal outcomes. However, the complexity of modern operations, characterized by dynamic environments, varying demands, and resource constraints, makes scheduling a formidable challenge.

One of the primary challenges in scheduling is dealing with uncertainties and variability. Unpredictable factors such as machine breakdowns, supply chain disruptions, and fluctuating customer demands can derail even the most meticulously planned schedules.

In addition, the need to balance multiple, often competing objectives—such as minimizing costs, maximizing resource utilization, and meeting deadlines—adds another layer of complexity to the scheduling process.

2.3 Industry 4.0 and the growing importance of scheduling

The advent of Industry 4.0 has significantly heightened the importance of scheduling in modern industries.
These technologies enable real-time data collection and analysis, which can be leveraged to optimize scheduling decisions dynamically. For example, IoT sensors can monitor equipment status in real-time, allowing for predictive maintenance and reducing unexpected downtimes [29]. AI algorithms can analyze vast amounts of data to predict demand patterns and adjust production schedules, accordingly, ensuring that supply meets demand efficiently [33].

The interconnected nature of Industry 4.0 systems means that scheduling must be adaptive and capable of responding to real-time changes in the production environment. This adaptability is crucial for maintaining operational efficiency and competitiveness in a rapidly changing market. Furthermore, the ability to integrate sustainability considerations into scheduling processes is enhanced by the advanced data analytics capabilities of Industry 4.0 technologies. This integration supports more sustainable production practices, aligning economic and environmental goals.

3 Job shop scheduling problem

3.1 Problem overview

The job shop scheduling problem (JSSP) is a widely studied combinatorial optimization problem in the operational research and management field.

The term scheduling refers to the process of determining the timing and sequence of tasks or activities to be completed within a specific timeframe or resource constraints.

The classical form of the problem is known as the deterministic job shop scheduling problem (DJSSP) [40].
The DJSSP consist of a finite set of *n* jobs, denoted as $J = \{J_1, J_2, \dots, J_n\}$, and a finite set of *m* machine, denoted as $M = \{M_1, M_2, \dots, M_m\}$.
Each job J_k involves an ordered sequence of m_k operations, denoted as $O = \{O_{k1}, O_{k2}, \dots, O_{km_k}\}$, with each operation to be executed on a machine. These operations are characterized by their start time, denoted as t, and processing time, denoted as τ.

The problem incorporates capacity constraints, meaning that each machine can only handle one operation at a time, and each job can only be processed on one machine at a time.
The dimensionality of the problem instance is *n x m*.

The general definition of the problem theoretically allows for jobs to be processed more than once on the same machine (machine repetitions) or not to be performed on a machine during its processing steps (machine absence). However, for simplicity, it is often assumed that each job is processed exactly once on each machine, in this case the dimensionality of the problem equals *nm*.

The resolution of the problem requires finding the optimal operation sequence in relation to an objective function.
Some common objective functions [41], include:

- Makespan minimization. “The makespan is the completion time of the last job to leave the system.” [42]
- Total weighted completion time minimization. The weighted completion time represents the total holding or inventory cost incurred by the schedule.
- Maximum of lateness minimization. The lateness relative to a job is the difference between its completion time and its due date which is positive in case the job is completed late and negative in case the job is completed early.

Despite there being several theoretical objective functions that embody different industrial criteria, the most widely used objective function in academic research is the minimization of the maximum makespan C_{max}.

The makespan can be formally expressed as follow:

$$C_{max} = \max(t_{ik} + \tau_{ik}): \qquad \forall\, J_t \in J, \qquad m_k \in M$$

Its adoption is justified by an ease in the problem formulation and its mathematical handling. [1]

Table 1 shows an example of deterministic job shop problem instance from Muth and Thompson (1963) [43].
The ft06 job shop scheduling problem is one of the most well-known benchmarks in operations research and involves scheduling six jobs on six machines.
The dimensionality of the problem is 6 x 6.
The table defines the processing steps associated to each job specifying the machine number and the processing time of each operation.

Table 1: Example of Deterministic Job Shop Scheduling Problem (problem instance FT06)

Job	*Operations*					
J1	M3, 1	M1, 3	M2, 6	M4, 7	M6, 3	M5, 6
J2	M2, 8	M3, 5	M5, 10	M6, 10	M1, 10	M4, 4
J3	M3, 5	M4, 4	M6, 8	M1, 9	M2, 1	M5, 7
J4	M2, 5	M1, 5	M3, 5	M4, 3	M5, 8	M6, 9
J5	M3, 9	M2, 3	M5, 5	M6, 4	M1, 3	M4, 1
J6	M2, 3	M4, 3	M6, 9	M1, 10	M5, 4	M3, 1

The solution to the ft06 problem is showed in the figure below.

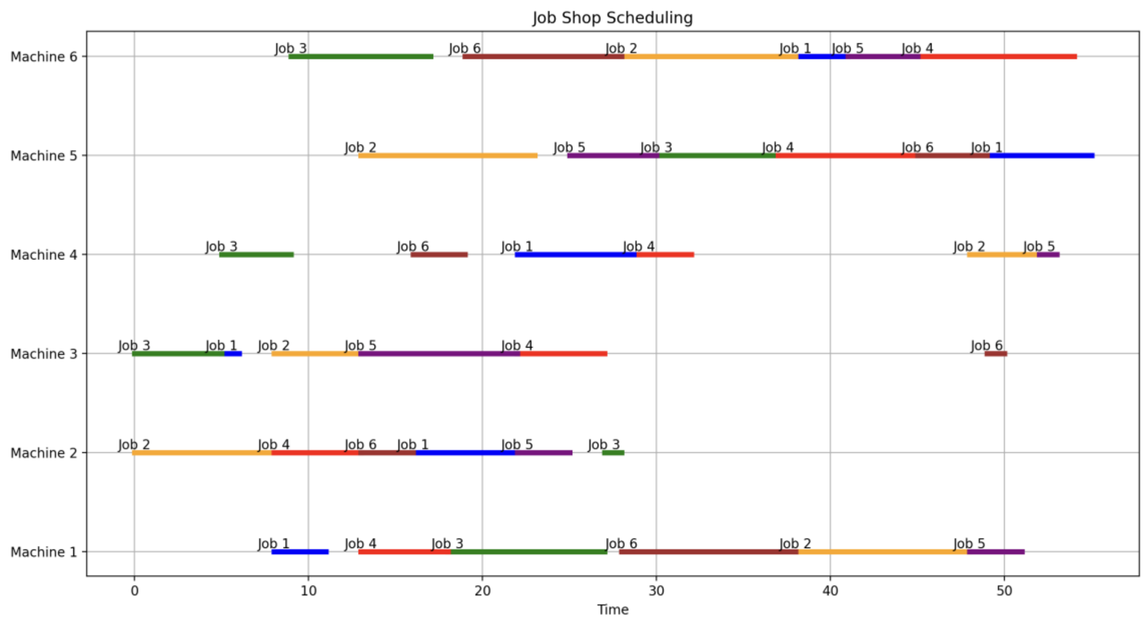


Figure 2: ft06 job shop scheduling solution

3.2 Problem variants and classification

The strong connection of the job shop scheduling problem with the real-world manufacturing and production environments has led researchers to explore several variants of the problem with increasing complexity and uncertainty.
The deterministic Job shop Scheduling problem, indeed, can be further complicated with different combinations of constraints and problem settings to address all possible real-world scenarios effectively.

According to the [41] 14 different classes of JSSP can be listed. The most relevant to this thesis are the following:

- The deterministic JSSP defined above which can be characterized by different machine environments including single machine model, parallel machine model, flow shop and job shop [42].

- Static JSP refers to a scheduling problem that is formulated and solved assuming that there no changes in parameters or inputs during the scheduling process will occur. Static JSPs are common in scenarios where the scheduling horizon is short or where there is little variability in the system parameters over time.

- Dynamic JSP in which the scheduling environment is subject to changes over time including variations in job arrival times, processing times, machine breakdowns, resource availability and job priority.

All the other problem variants can be consulted in the figure below.

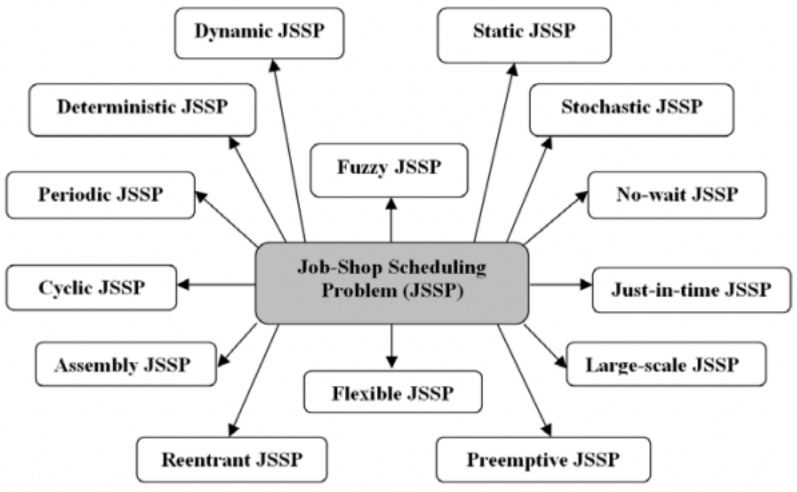


Figure 3: Fourteen classes of JSP according to [3]

3.3 Problem representation and classification

The primary classification method for scheduling problems is the four-field notation (A/B/C/D) introduced by Conway in 1967 [46], where A is the number of jobs, B the number of machines, C the flow pattern within the machine shop and D is the performance measure for evaluating the schedule.

The Job Shop Scheduling Benchmark Problem ft06 represented in Table 1, using Conway notation might be expressed as follows:

$$(6/6/1/C_{max})$$

where 1 indicates a single flow pattern within machine shop.

In 1964 Roy and Sussmann [47] developed a model to represent and solve the JSSP called disjunctive graph model.
A disjunctive graph model is composed of:

- A set N of nodes representing the operations to be performed on the set M of machines. Two fictious operations, that represents the initial and final stage of the schedule, are added to the set N.

- A set E of direct conjunctive edges representing the precedence constraints between operations. The edge (i, j) indicates that operation i must be complete before the beginning of operation j.
  The edge weight corresponds to the processing time of the starting node's operation.

- A set D of disjunctive edges represent the capacity constraint of the problem, ensuring each machine to process only one job at a time.

A disjunctive graph representing the connection of the first stage of the Job Shop Scheduling Benchmark Problem FT06 is shown in the following Figure:

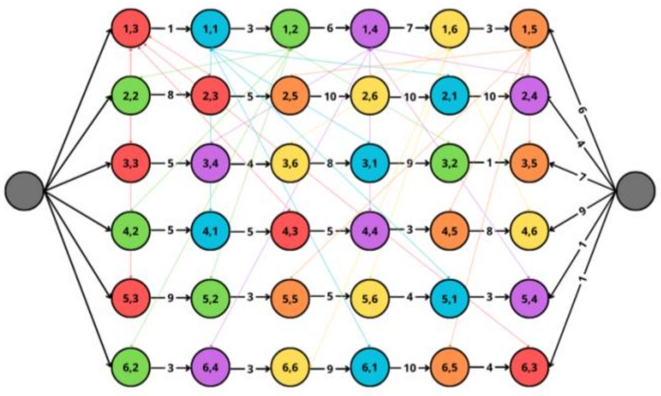


Figure 4: Disjunctive graph model ft06: first layer

It is noticeable that the number of possible solutions of the problem, represented by the colored lines, increases exponentially with the number of jobs and machines in the problem definition.

3.4 Problem history

The Job Shop Scheduling Problem has its roots in the operational research and industrial engineering filed [40].

Even though it is challenging to pinpoint the exact time the problem was first proposed, it started gaining significant importance during the mid-20th century when researchers began exploring methods to optimize production processes using innovative techniques and algorithms.

One of the earliest contributions to the field was Johnson's two-machines flow shop resolution algorithm in 1954 [44], which was later adapted to address the JSSP by Jackson in 1956 [45].

Throughout the 1950s, research was mostly focused on the definition and application of heuristic methods: methods that prioritize the research of a satisfying solution in a reasonable amount of time rather than guaranteeing an optimal solution of the problem instance.
In the 1960s attention shifted towards enumerative algorithm based on more accurate and complex mathematical constructs to achieve exact solutions to the problem instances.

During the 1970s and 1980s, research on job shop scheduling highlighted the problem complexity leading to the designation of two eras: "Before Complexity" (BC) and "Advanced Difficulty" (AD).

During the late 1980s and early 1990s, significant progress was made in solving the job shop scheduling problem with the development of innovative metaheuristic algorithms.
Unlike traditional algorithms, which are designed for specific problem instances, metaheuristic algorithms provide general frameworks for finding approximate solutions to optimization problems.

In the late 1990s, several iterative methods including artificial intelligence and neural networks methods became increasingly important in addressing the complexity of the JSSP.

In recent years, the Job Shop Scheduling Problem has continued to be an active area of research, with advancements driven by both theoretical developments and practical applications.

3.5 Problem relevance

The importance of the Job shop Scheduling Problem in the academic field arises from its inherent characteristics, the principal being its complexity that creates a fertile ground for developing and testing optimization algorithms and heuristics methodologies.

As the number of jobs and machine increases, the potential number of feasible schedules grows exponentially. The exponential growth makes finding an optimal solution computationally challenging, particularly for large-scale instances.

JSSP is classified as NP-hard, meaning that it belongs to a class of computationally difficult problems for which no polynomial-time algorithm exists to guarantee finding the optimal solution. This complexity arises from the need to explore a vast search space to identify the best schedule among all possible combinations. [42]

The complexity of the JSSP is further compounded by its various problem variants introducing additional constraints like setup times, release dates, and due dates. Each constraint introduces additional intricacies that increase the difficulty of finding optimal solutions.

Another key element of the JSSP is its real-world relevance across various industries.

In manufacturing industries such as automotive, electronics, and aerospace, efficient job scheduling is crucial for optimizing production processes. By determining the sequence of tasks and allocating resources (machines, tools, and manpower) effectively, manufacturers can minimize production time, reduce costs, and enhance overall productivity.

In logistics and transportation sectors, the JSSP finds applications in scheduling vehicles, routes, and deliveries. Efficient scheduling ensures timely delivery of goods, optimal utilization of transportation assets, and cost-effective operations.

In healthcare settings, scheduling plays a vital role in managing patient appointments, allocating medical resources, and optimizing staff schedules. Healthcare providers use scheduling algorithms to minimize patient waiting times, improve resource utilization, and enhance the quality-of-care delivery.

The JSSP also has applications in service industries such as call centers, utilities management, and facility maintenance. Efficient scheduling of service appointments, maintenance tasks, and workforce assignments helps organizations deliver timely and reliable services to customers. By optimizing scheduling processes, service providers can improve customer satisfaction, minimize service downtime, and maximize operational efficiency.

3.6 Resolution methodologies

Several approaches and algorithms have been developed over time to tackle the Job Shop Scheduling Problem.
Resolutions methods can be grouped into four main categories:

- Exact methods.

  Exact methods refer to algorithms or approaches that aim at finding the optimal solution of the problem exhaustively exploring the entire solution space or using rigorous mathematical formulations.

Exact methods ensure optimality but may suffer from scalability issues as the problem size increases due to the exponential growth of the solution space.
Two of the most important exact methods are:

- Branch and Bound (B&B): This method systematically explores all possible schedules to find the optimal one. It uses upper and lower bounds to prune the search space and eliminate suboptimal solutions early. Despite its accuracy, B&B is computationally intensive and may not be practical for larger instances of the job shop problem. [48]
- Integer Linear Programming (ILP): ILP formulations translate the scheduling problem into a set of linear equations and inequalities, which can then be solved using optimization solvers like CPLEX [50] or Gurobi [51]. While powerful, ILP can become computationally prohibitive for complex problems. [49]

- Heuristic methods.

In situations where funding the optimal solution is computationally infeasible, heuristic methods aim at finding a satisfying solution in a reasonable period of time.
The most used heuristic method bases its efficiency upon the priority dispatching rules.
Priority Dispatching Rules: These simple, rule-based methods prioritize jobs based on specific criteria, such as shortest processing time (SPT), earliest due date (EDD), or critical ratio (CR). While not guaranteed to find the optimal solution, they are quick and easy to implement.

- Metaheuristic methods.

Metaheuristic methods aim to efficiently explore large solution spaces to find near-optimal solutions within a reasonable amount of time.
Metaheuristic methods include:

- Genetic Algorithms (GA): GAs simulate the process of natural evolution, using selection, crossover, and mutation operators to evolve a population of solutions over successive generations. This method is effective for exploring large search spaces and finding near-optimal solutions. [52]
- Simulated Annealing (SA): Inspired by the annealing process in metallurgy, SA searches for a good solution by allowing occasional uphill moves to escape local optima, gradually reducing the likelihood of such moves as the search progresses. [53]

- Other methodologies.

Starting from the late 1990, different techniques were implemented for the resolution of the JSSP, such as machine learning, reinforcement learning, and neural network techniques can be adopted as a resolution method.

The main resolution methods belonging to each category are reported in Table 2.

Table 2: JSSP resolution methods

Category	Resolution methods
Exact methods	Branch and Bound (B&B) Branch and Cut (B&C) Branch and Price (B&P) Integer Linear Programming (ILP) Constraint Programming (CP)
Heuristic methods	Priority Dispatching Rule (PDR)
Metaheuristic methods	Local search (LS) Tabu search (TB) Genetic algorithms (GAs)
Other methodologies	Machine learning Reinforcement learning Multi-agent systems Neural networks Hybrid methods

3.6.1 Priority Dispatching Rules

Priority dispatching rules are fundamental to job shop scheduling, where the objective is to determine the optimal sequence of jobs processed on various machines.
These rules are crucial in enhancing efficiency, reducing wait times, and optimizing resource utilization in manufacturing and service industries [54].

The Longest Processing Time (LPT) rule prioritizes jobs with the longest duration, aiming to keep the most critical machines busy and minimize idle time, which can be particularly beneficial in environments with high machine utilization rates.

Conversely, the Shortest Processing Time (SPT) rule focuses on minimizing the total job completion time by prioritizing jobs with the shortest duration, which helps in reducing the average job flow time and improving throughput.

The First-In-First-Out (FIFO) rule, also known as First-Come-First-Served, processes jobs in the order they arrive, ensuring fairness and simplicity but potentially leading to inefficiencies if early jobs are significantly long.

The Last-In-First-Out (LIFO) rule, which processes the most recently arrived jobs first, can be effective in scenarios where newer jobs are more critical, but might cause older jobs to experience excessive delays, potentially increasing the average job tardiness [42].

Each of these rules has distinct advantages and limitations, making their application context-dependent and often necessitating a combination of rules for optimal job shop performance [55].

3.6.2 Genetic algorithm

Genetic Algorithms (GAs) are metaheuristic optimization techniques inspired by the principles of natural selection and evolution.
Introduced by John Holland in the 1960s, GAs have become a popular approach for solving complex optimization problems across various domains [56]. The fundamental idea behind GAs is to evolve a population of potential solutions (individuals) over multiple generations, applying genetic operators such as selection, crossover, and mutation to improve the quality of solutions iteratively.

The basic structure of a genetic algorithm includes the following components [52]:

- Initialization: Create an initial population of potential solutions.
- Fitness Evaluation: Assess the quality of each solution using a fitness function.
- Selection: Choose individuals for reproduction based on their fitness.
- Crossover: Combine genetic information from selected parents to create offspring.
- Mutation: Introduce random changes in individuals to maintain genetic diversity.
- Replacement: Form a new population from the offspring and (optionally) some parents.
- Termination: Repeat steps 2-6 until a stopping criterion is met.

3.6.2.1 Job Shop Scheduling Genetic Algorithm

For this thesis, a specific genetic algorithm has been developed to address the dynamic job shop scheduling problem. Some of the most important functions are reported in the Appendix.

Key components of the implemented GA for JSSP:

- Representation: The algorithm uses a permutation-based encoding, where each individual (chromosome) represents a sequence of job operations. This representation ensures that precedence constraints within jobs are always satisfied.
- Population Initialization: The initial population is created by generating random permutations of job operations, ensuring diversity in the starting solutions.
- Fitness Function: The fitness of an individual is determined by the makespan the schedule it represents. The algorithm aims to minimize this makespan.
- Selection: Individuals are selected for reproduction using a fitness-proportionate selection method.

- Crossover: A single-point crossover operator is used to create offspring from selected parents. The crossover point is randomly chosen, and genes are inherited from both parents while maintaining schedule validity.

- Mutation: The mutation operator randomly swaps two positions in the chromosome, introducing small changes that help maintain genetic diversity and explore the solution space.

- Evolution: The population evolves over a specified number of generations. In each generation, selection, crossover, and mutation operations are applied to create a new population.

- Schedule Decoding and Execution: The best individual from the final generation is decoded into a feasible schedule. Operations are scheduled according to the order specified in the chromosome, respecting machine availability and job precedence constraints.

- Dynamic Job Arrival: The algorithm is adapted to handle dynamic job arrivals, simulating a more realistic production environment where new jobs enter the system over time. As new jobs arrive, they are added to the problem set and GA is re-run to schedule the remaining operations.

- Processing Time Variability: To account for uncertainties in real-world scenarios, the algorithm incorporates variability in processing times, reflecting potential fluctuations in machine performance or job complexity.

This genetic algorithm implementation for JSSP combines elements from classical GA approaches with problem-specific adaptations to address the complexities of job shop scheduling. The integration of dynamic job arrivals and processing time variability enhances the algorithm's applicability to real-world manufacturing scenarios, where uncertainty and changing conditions are common challenges.

While this algorithm is only inspired by a major publication [57], its comprehensive approach to simulating and solving dynamic job shop problems makes it a valuable benchmark for comparing other scheduling algorithms. It incorporates key elements of modern scheduling research, such as dynamism and uncertainty, which are often overlooked in classical benchmark problems.

4 Reinforcement learning

4.1 Artificial Intelligence and Machine Learning introduction

Stanford Professor John McCarthy who first used the term “Artificial Intelligence” in 1955, defines it, in a subsequent article [58], as “the science and engineering of making intelligent machines, especially intelligent computer programs”.

AI is a multidisciplinary field comprising different subfields such as machine learning, natural language processing, computer vision and more.
Among them machine learning (ML) focuses on the development of algorithms and models that enable computers to learn patterns and relationships from large amounts of data and use that knowledge to and make predictions or decisions without being explicitly programmed.

Machine learning models can be grouped into three primary categories [59]:

- Supervised machine learning: in supervised learning, the algorithm is trained on a labeled datasets in which input data are associated with corresponding output labels. The goal is for the algorithm to be able to accurately classify new, unseen data or predict outcomes.
  Supervised learning algorithm’s applications include predictive analytics, object detection and classification.

- Unsupervised machine learning: in unsupervised learning, the goal of the algorithm is to analyze and cluster unlabeled dataset to discover hidden patterns or data groupings.
  Unsupervised learning algorithm’s applications include recommendation systems, customer segmentation and big data visualization.

- Reinforcement learning: reinforcement learning is similar to supervised learning but is well-suited for tasks where explicit supervision is unavailable due to the lack of labeled data set. The algorithm is trained through trial-and-error mechanism to learn a policy or strategy that maximizes its reward.

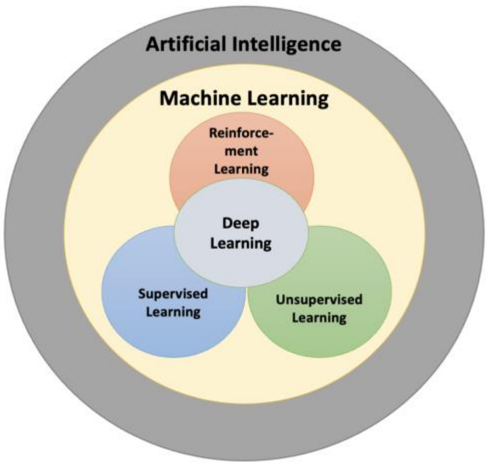


Figure 5: Artificial Intelligence paradigms

4.2 Reinforcement learning overview

Reinforcement Learning (RL) is a machine learning paradigm to solve sequential decision-making problem [60].
It involves an agent which learns to make decisions by performing certain actions in an environment to maximize cumulative reward. Unlike supervised learning, which requires labeled input-output pairs, RL is based on the interaction between the agent and the environment, using trial and error to discover optimal actions.

The fundamental elements of an RL system include:

- Agent: The learner or decision-maker that interacts with the environment.
- Environment: The external system with which the agent interacts and receives feedback.
- State (S): A representation of the current situation of the environment.
- Action (A): The set of all possible moves the agent can make.
- Reward (R): Feedback from the environment to evaluate the action taken.
- Policy (π): The strategy used by the agent to determine the next action based on the current state.
- Value Function (V): Estimates the expected cumulative reward from a given state.
- Q-Function (Q): Estimates the expected cumulative reward from a given state-action pair.

The underlying functioning of this method involves an agent interacting, over time, with an environment to achieve a long-term goal.
The agent-environment interaction, represented in Figure 5, can be explained as follow:

```
at each time step t:
    the agent observes the environment's state $S_t$ ,
    the agent selects an action $A_t$ to be performed on the environment,
    the action is performed,
    the agent receives a reward $R_t$ (reinforcement),
    the environment transitions to a successor state $S_{t+1}$ .
```

The sequence above is repeated until a terminal state is reached marking the end of a training episode.

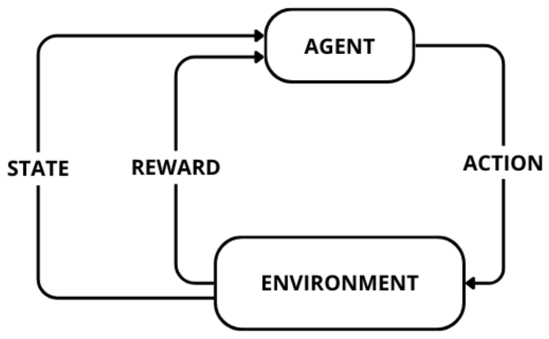


Figure 6: Reinforcement learning functioning [60]

RL operates based on the principle of learning from interaction. The process can be summarized as follows:

- Initialization: Initialize the policy and value function.
- Interaction: The agent observes the current state S_t.
- Action Selection: The agent selects an action A_t based on the current policy π^*.
- Feedback: The environment responds to the action with a reward R_t and transitions to a new state S_{t+1}.
- Update: The agent updates its policy and value function based on the reward and new state.
- Iteration: Repeat the process for a predefined number of episodes or until convergence.

The agent chooses the action to perform based on a policy π_t. The policy expresses the probability the agent will take a specific action (among the available ones) in a specific state.
The agent discovers which action yields the most reward through a trial-and-error search in which each action is rewarded based on both the immediate and delayed reward.
The agent's policy is updated throughout the learning phase as a result of the agent experience.
The ultimate goal is to find the optimal policy π^* that maximizes the expected cumulative reward from any given state.
Each type of reinforcement learning algorithm updates the agent policy differently based on the agent experience.

4.3 Markov Decision Processes

The Markov Decision Process (MDP) is a mathematical framework to model sequential decision-making problems including basic reinforcement learning problems [61].
To be more precise a problem can be considered a Markov Decision Process if it satisfies the Markov property that is, if the transition from state s to state s' depends only upon the state s.

A Markov Decision process model is composed of five elements:

- Decision epochs: in a Markov decision process, decisions are made at points in time referred to as decision epochs.

- States: at each point in time the environment occupies a state. The set of all possible states is called state space s.

- Actions: at each point in time the agent can undertake an action. The set of all available actions available is called action state a.

- Transition probabilities: the state the environment transitions to, as a result of an action at the decision epoch t, is determined by the transition probability function which gives the probability of the next state and reward given the current state and action. In the basic versions of the Markov Decision Processes, the transition probability from one state to another is equal to one.

- Rewards: the decision maker receives a reward on the basis of the action undertaken in a specific state at a specific decision epoch.

In reinforcement learning, the agent learns to make decisions by interacting, over time, with an environment described by a Markov Decision Process.

The goal is for the agent to maximize the expected sum of future rewards it receives over time. This sum is represented by:

$$R_t = \sum_{k=0}^{\infty} \gamma^t \, r(t,k)$$

where t is the current time step, k represents the time step into the future at which the reward is received, $r(t+k)$ is the immediate reward received at time $(t+k)$ and γ (gamma) is a discount factor that controls the importance of future rewards relative to immediate ones.

The value $V^{\pi}(s)$ of being in a particular state s under a given policy π is the expected sum of future rewards the agent can expect to receive from that state onward.
It is obtained by summing over all possible actions a that can be taken from state s, and for each action, summing over all possible next states s' that can be reached from state s after taking action a.

For each next state s', the value function considers the immediate reward $R^a_{ss'}$ obtained by taking action a and transitioning to state s', plus the discounted value $\gamma V^\pi(s')$ of being in state s' onwards under policy π.

This calculation accounts for the expected rewards and their discounting over time.

$$V^\pi(s) = \sum_{a \epsilon A} \pi(s,a) \sum_{s' \epsilon S} (R^a_{ss'} + \gamma V^\pi(s'))$$

The formula above is called Bellman equation for the value function. It expresses the relationship between the value of the state s and the values of the following states reached following a specific policy.

The value $Q^\pi(s,a)$ of taking a particular action a in a given state s under policy π is the expected sum of future rewards the agent can expect to receive by taking that action and then following the policy.
Similar to V^π, it considers the immediate reward $R^a_{ss'}$ obtained by taking action a and transitioning to a next state s', plus the discounted value $\gamma V^\pi(s')$ of being in state s' onwards under policy π.

$$Q^\pi(s,a) = \sum_{s' \epsilon S} (R^a_{ss'} + \gamma V^\pi(s'))$$

These value functions, V^π for states and Q^π for state-action pairs, help the agent make decisions by estimating the long-term rewards associated with different states and actions. They guide the agent towards actions that lead to higher cumulative rewards over time.

During learning, the agent aims to find an optimal policy π^* that outperforms all other policies in terms of accumulating maximum rewards.
It has been mathematically proven that for every Markov Decision Process (MDP), there exists an optimal policy π^*.

The optimal policy $\pi^*(s)$ is determined by selecting the action that maximizes the sum over all possible next states s' of the immediate reward $R^a_{ss'}$ obtained by taking action a and transitioning to state s', plus the discounted value $\gamma V^\pi(s')$ of being in state s' onwards.

$$\pi^*(s) = \max_{a \epsilon A} \{ \sum_{s' \epsilon S} \left(R^a_{ss'} + \gamma V^\pi(s') \right) \}$$

Dynamic programming refers to the algorithm used to find the optimal policy given complete knowledge of the environment.

4.4 Types of Reinforcement Learning

RL can be broadly categorized into different classes: [60]

1. Model-Free vs. Model-Based RL.

    - Model-Free RL algorithms learn directly from experience without explicitly modeling the environment. They are generally more flexible but may require more samples to learn effectively. Examples include Q-learning and SARSA.
    - Model-Based RL algorithms learn a model of the environment, which allows for planning and potentially more sample-efficient learning. However, they can be sensitive to model errors. Examples include Dyna-Q and PILCO

2. Value-Based vs. Policy-Based RL.

    - Value-Based methods estimate the value of being in a particular state or taking a specific action and derive a policy from these estimates. Q-learning and Deep Q-Networks (DQN) are prominent examples.
    - Policy-Based methods directly optimize the policy without maintaining a value function. They can be more effective in continuous or high-dimensional action spaces. REINFORCE and PPO (Proximal Policy Optimization) are examples of this approach.

3. On-Policy vs. Off-Policy RL.

    - On-Policy methods use the same policy for both behavior generation and policy improvement. They can be more stable but less sample efficient. Examples include SARSA and PPO.
    - Off-Policy methods can learn from data generated by a different policy, potentially improving sample efficiency. Q-learning and Deep Deterministic Policy Gradient (DDPG) are off-policy algorithms

4.5 Principal Reinforcement Learning Algorithms

4.5.1 Q-learning

The Q-learning is a popular off-policy reinforcement learning algorithm.[60] [62]

The core of the algorithm is the iterative update of a Q-table associating each state–action pair $(s^t, a^t)$to a value called Q-value which is updated on the basis of the observed reward and the discounted maximum Q-value of the next state. The Q-table is usually initially set to zero.

At each time step, the agent chooses an action to take using the epsilon – greedy policy.
The epsilon greedy policy is a strategy used to balance the exploration and exploitation during the learning process. In the Q-learning algorithm it is usually implemented in the following way:

- With probability ε, the agent selects a random action (exploration)
- With probability $1 - \varepsilon$, the agent selects the action with the highest Q-value for the current state (exploitation)

The parameter ε is typically set to decrease over time, reflecting the idea that as the agent learns more about the environment, it should rely more on exploitation and less on exploration.

Once the action is selected, the Q-value of the state is updated.
The Bellman function that updates the Q-values can be expressed as follows:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

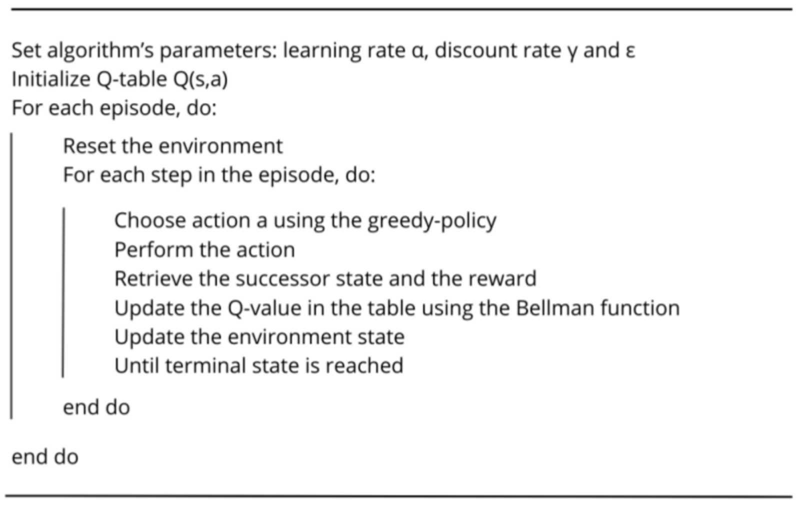
Where α represents the learning rate determining the extent to which new information overrides old information, r_t represents the immediate reward received after taking the action a in state s, γ I the discount factor representing the importance of future rewards.

In the Q-learning algorithm, the Bellman equation is used to update the Q-values after each action is taken, gradually refining the agent's understanding of the environment and improving its decision-making capabilities.

The process is repeated and the Q-values are updated until the agent reaches a terminal state or a predefined number of iterations.

With sufficient exploration and appropriate learning rate and discount factor settings, Q-learning converges to the optimal Q-values, ensuring that the agent learns the optimal policy for maximizing cumulative rewards over time.

The basic steps of the Q-learning algorithm are shows in the following image:



```
Set algorithm's parameters: learning rate α, discount rate γ and ε
Initialize Q-table Q(s,a)
For each episode, do:
    Reset the environment
    For each step in the episode, do:
        Choose action a using the greedy-policy
        Perform the action
        Retrieve the successor state and the reward
        Update the Q-value in the table using the Bellman function
        Update the environment state
        Until terminal state is reached
    end do
end do
```

Figure 7: Q-learning algorithm steps

5 Deep Reinforcement Learning

5.1 Deep reinforcement learning overview

Deep reinforcement learning (DRL) is a subfield of artificial intelligence that combines reinforcement learning techniques with deep learning methods, specifically deep neural networks.
It involves training AI agents to learn optimal behaviors by interacting with an environment using deep neural networks to approximate the agent's policy (the mapping from states to actions) and/or value functions (estimating the expected future rewards).

These networks enable RL agents to handle high-dimensional state and action spaces, making them suitable for a wide range of real-world applications.
DRL includes several algorithms such as deep Q-networks (DQN), policy gradient methods (such as Proximal Policy Optimization), actor-critic architectures, and more.

5.2 Deep learning

Deep learning, a specialized subset of machine learning, has significantly transformed the landscape of artificial intelligence (AI) by enabling the modeling of intricate patterns and complex representations through deep neural networks [63].
Deep learning, indeed, uses algorithms, called Deep Neural Networks (DNN), whose functioning is inspired by the structure of the neural networks in human brains. The purpose of deep learning is trying to mimic how the human brain behaves through multi-layers neural networks.

The foundation of deep learning lies in its capability to automatically learn and extract features from raw data, which traditional machine learning algorithms often struggle to achieve. This paradigm shift has facilitated the development of models that excel in various complex tasks, achieving human-level or even superhuman performance in some cases.

The inception of deep learning dates back several decades, but its true potential was realized in the 21st century with advancements in computational power, availability of large datasets, and innovative algorithms. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, often referred to as the "godfathers" of deep learning, have been pivotal in this field's progress, contributing foundational theories and practical implementations that have underpinned modern AI systems [64].

The application of deep learning has led to groundbreaking advancements in several domains:

- Image Recognition: Deep convolutional neural networks (CNNs) have set new benchmarks in image classification, object detection, and segmentation tasks, surpassing human-level accuracy in some benchmarks [65].

- Natural Language Processing (NLP): Recurrent neural networks (RNNs), Long Short-Term Memory (LSTM) networks, and transformers have revolutionized NLP tasks such as machine translation, sentiment analysis, and language generation, enabling applications like real-time translation and conversational agents [66].

- Game Playing: Deep reinforcement learning has enabled AI systems to master complex games such as Go, chess, and various video games. Notable examples include AlphaGo and AlphaZero, which have demonstrated strategic thinking and planning capabilities previously thought to be exclusive to humans [67].

5.2.1 Fundamental Concepts of Deep Learning

Deep learning involves training neural networks with many layers, known as deep neural networks (DNNs). Unlike traditional machine learning algorithms that rely on manual feature extraction, deep learning models automatically learn hierarchical representations from raw data. The key idea is that each layer in a DNN captures increasingly abstract features, enabling the model to understand complex patterns in the data.[63]

The primary components of a neural network include: [68]

- Neurons.

  Neurons are basic processing units that receive inputs, apply a non-linear transformation, and produce an output. Each neuron performs a weighted sum of its inputs followed by an activation function to introduce non-linearity.

- Layers:
  - Input Layer: The first layer of the network, which receives raw input data. Each neuron in this layer represents a feature from the input data.
  - Hidden Layers: Intermediate layers between the input and output layers. These layers perform feature extraction and transformation. The depth (number of hidden layers) and width (number of neurons per layer) of a network can vary based on the complexity of the task.
  - Output Layer: The final layer that produces the network's predictions or classifications. The number of neurons in this layer typically corresponds to the number of classes in a classification task or the number of outputs in a regression task.

- Weights and Biases.

  Parameters that determine the strength of connections between neurons. Weights are adjusted during training to minimize the difference between the network's predictions and the actual outputs. Biases allow the activation function to be shifted to the left or right, which can be critical for learning.

- Activation Functions.
  Non-linear functions applied to the input of each neuron to introduce non-linearity, enabling the network to learn complex patterns. Common activation functions include the sigmoid function, hyperbolic tangent (tanh), and rectified linear unit (ReLU).

- Loss Function.

  A function that measures the difference between the predicted output and the true output, guiding the training process. Common loss functions include mean squared error (MSE) for regression tasks and cross-entropy loss for classification tasks.

- Optimizer.

  An algorithm that adjusts the network's weights and biases to minimize the loss function. Optimizers use gradient-based methods such as stochastic gradient descent (SGD), Adam, and RMSprop to update the parameters in the direction that reduces the loss.

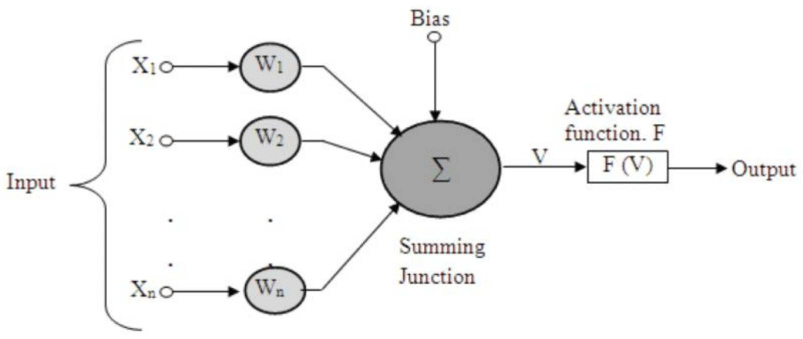


Figure 8: Single neuron with its elements [68]

The functioning of a neural network can be divided into two main phases: forward propagation and backpropagation.

- Forward Propagation

  During forward propagation, input data passes through the network layer by layer. Each neuron's output is computed as follows:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$a = \sigma(z)$$

where z is the weighted sum of inputs x_i with weights w_i and bias b, and σ is the activation function. The output a is then passed as input to the next layer. This process continues until the final output is produced.

- Backpropagation

  Backpropagation is used to update the weights and biases based on the error calculated from the loss function.
  It involves the following steps:

  - Computing the Loss: The difference between the predicted output and the actual output is calculated using the loss function.

  - Calculating Gradients: Using the chain rule, the gradients of the loss with respect to each weight and bias are computed. These gradients indicate the direction and magnitude of change needed to minimize the loss.

  - Updating Parameters: The weights and biases are adjusted in the direction that reduces the loss, typically using an optimization algorithm like stochastic gradient descent (SGD).

5.2.2 Types of Neural Networks

Several types of neural networks have been developed to address different tasks and data types. The most notable ones include:

- Feedforward Neural Networks (FNN) [68]

  Feedforward neural networks, also known as multi-layer perceptron (MLPs), are the simplest type of artificial neural networks. They consist of an input layer, one or more hidden layers, and an output layer, with connections moving in one direction from input to output.

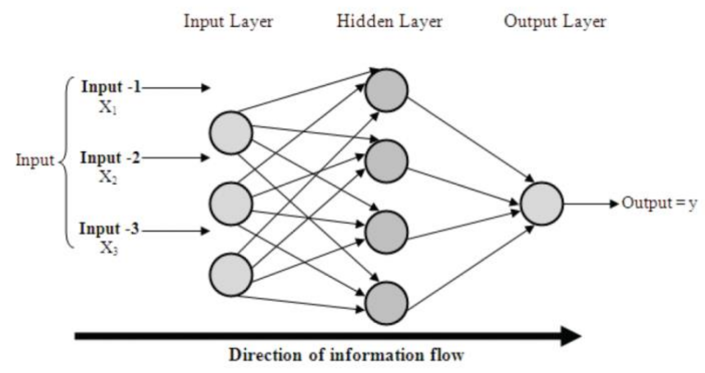


Figure 9: Feedforward Neural Network: Information flows in one direction from the input layer through hidden layers to the output layer [68]

- Convolutional Neural Networks (CNN)

  Convolutional neural networks are designed for processing structured grid data, such as images. CNNs use convolutional layers to automatically learn spatial hierarchies of features through local connections and shared weights, followed by pooling layers to reduce dimensionality (LeCun et al., 1998).
  Elements of CNNs include:
  - Convolutional Layers: Apply filters to the input to create feature maps.
  - Pooling Layers: Reduce the spatial dimensions of the feature maps.
  - Fully Connected Layers: Combine features to make the final prediction.

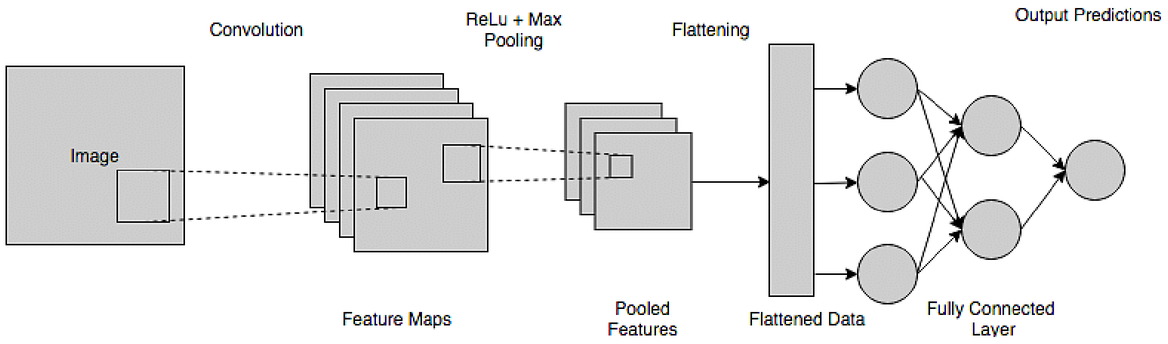


Figure 10: Convolutional Neural Network: Consists of convolutional, pooling, and fully connected layers [69]

5.3 Deep Reinforcement Learning Algorithms

5.3.1 Deep Q-learning

Deep Q-Learning (DQN) is an extension of Q-Learning that addresses the limitation of handling large state-action spaces by using deep neural networks.
Instead of a Q-table, DQN utilizes a neural network to approximate the Q-function.
The neural network takes the state as input and outputs Q-values for all possible actions.

Another difference with respect to the Q-learning algorithm is the training method.
DQN incorporates experience replay, where experiences (state, action, reward, next state) are stored in a replay buffer. During training, batches of experiences are randomly sampled from the buffer to train the neural network. This helps in breaking correlations between consecutive experiences and stabilizes training.

DQN tends to perform better in complex environments with high-dimensional states compared to Q-Learning.

5.3.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a policy gradient method used to train AI agents to perform tasks. Policy Gradient Methods are among the most effective techniques in Reinforcement Learning. [70]

PPO objectives is for the agent to learn a policy that maps states to actions yielding the maximum possible expected sum of rewards.

The policy approximation is carried out using a neural network.
The policy network takes the current state as input and outputs a probability distribution over actions. During training, the parameters of this neural network are adjusted to improve the policy's performance.

The key innovation of PPO is the use of a clipped surrogate objective. Instead of directly optimizing the objective function, PPO constrains the policy update to be within a certain threshold. This helps to prevent large policy updates that may lead to instability.

Figure 11 shows a detailed process flow for a PPO reinforcement learning algorithm.

- Start: The process begins.
- Reset Simulation: The environment is reset to its initial state.
- Environment: This represents the problem space or world the agent interacts with.
- Actor: The agent that takes actions based on the current state.
- State and Action Loop:
    - The actor receives the state from the environment.
    - It then takes an action, which is fed back into the environment.
    - This loop continues until a terminal state is reached.
- RL PPO Library: Once observations are made, they're passed to the PPO algorithm implementation.
- Evaluate State: The current state is evaluated.
- Collect experience tuple: Information about the state, action, and reward is collected.
- Experience Collection Loop: Steps 6-8 repeat until the batch size is reached (exp < Batch_size).
- Select a minibatch: Once enough experience is collected, a minibatch is selected for training.

- Train policy using minibatch: The policy is updated using the selected minibatch.
- Batch Usage Check: The process checks if all batches have been used for training.
- Training Loop: If not all batches are used, it goes back to selecting another minibatch (step 10).
- Iteration Check: After training, it checks if the maximum number of steps (n_steps) has been reached.
- End or Continue:
    - If max steps are reached, the process ends.
    - If not, it goes back to resetting the simulation (step 2) for another iteration.

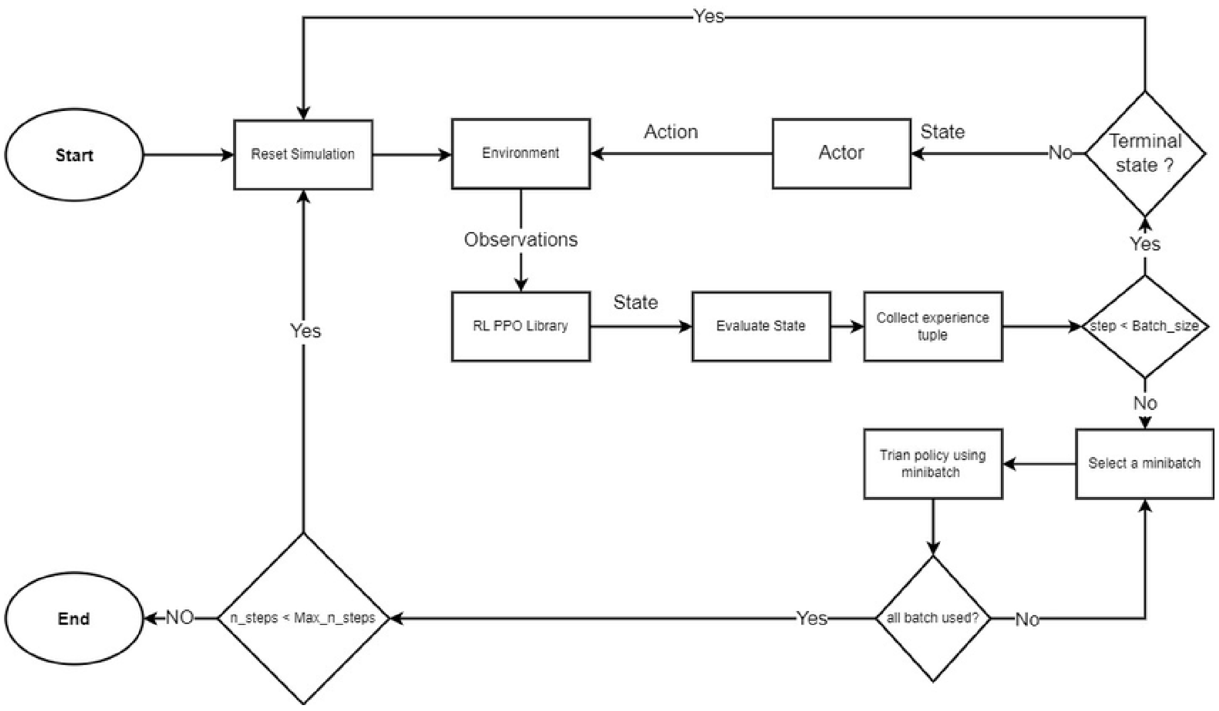


Figure 11: Proximal Policy Optimization flow chart [70]

7 Literature review

Reinforcement learning algorithms have become a significant focus in tackling job shop scheduling and its variants due to their ability to adapt and optimize in complex, dynamic environments. Deep reinforcement learning (DRL) approaches have shown promising results by leveraging neural networks to handle large state and action spaces effectively. This capability allows DRL to capture intricate patterns and dependencies within the scheduling problem, making it well-suited for real-world JSSP and DJSSP scenarios where the number of possible states and actions can be vast.

In the context of JSSP, researchers have explored several approaches leveraging different algorithms, state representations and reward functions to allow for stable learning approaches and long-term strategies exploitation.
Zhao et al. [72] introduced Q-learning as a foundational approach, focusing on iterative improvements through action-value updates based on observed rewards. Traditional reinforcement learning relies on value functions or policy representations, which can become unmanageable as scheduling problems increase in complexity. To address this issue, Tassel et al. [73] and Moon et al. [74] employed deep reinforcement learning methods, specifically proximal policy optimization and deep Q-network algorithms. DRL uses deep neural networks to process high-dimensional input data (such as job and machine states) and out- put optimal actions (such as job scheduling decisions) with increased efficiency and stability.

Given the dynamic nature of the environment, the dynamic job shop scheduling problem has been primarily tackled with deep reinforcement learning approaches to effectively manage the evolving nature of scheduling requirements. One prominent contribution by Zhao et al. [75] involves the adaptation of deep Q-networks, which utilize deep neural networks to approximate optimal action- value functions where the action set is composed of ten heuristic dispatching rules.
Another notable strategy exploited by Zhang et al. [76] includes Proximal Policy Optimization characterized by increased stability and fast optimization speed ensuring adaptation to changing scheduling conditions.

In this study, we propose an innovative approach to the Dynamic Job Shop Scheduling Problem utilizing Maskable Proximal Policy Optimization [77].
To achieve improved performances, the strategy proposed integrates the strengths of state-of-art approaches with several innovative features. The strategy proposed is based on PPO algorithm which is highly regarded for its stability, sample efficiency, and robust empirical performance in reinforcement learning.
To reduce the search space and enhance environment dynamicity, the action set has been designed to incorporate priority dispatching rules rather than considering each individual job and the use of masked actions ensures that the agent focuses on relevant and feasible decisions, enhancing the efficiency of the scheduling process.
Additionally, our approach employs an event-based control mechanism, where the agent is triggered only when necessary. This selective activation not only conserves computational resources but also allows for more responsive and adaptive scheduling adjustments.

8 Reinforcement learning algorithm

8.1 Algorithm Overview

The central core of the thesis work consists in the development and implementation of a reinforcement learning algorithm to tackle the job shop scheduling problem and its dynamic variant.
Job Shop Scheduling problem can be formulated as a sequential decision-making problem (or Markov Decision Process) and hence solved using Reinforcement Learning algorithm to train an agent to learn the optimal scheduling policy.

Possibilities to solve the JSSP using RL include both single-agent and multi-agent algorithms:

- Single – agent reinforcement learning: a single agent learns to make scheduling decisions for all the machines in the job shop environment,
- Multi-agent reinforcement learning: multiple agents (each representing a machine) learn to how to make scheduling decisions cooperatively or competitively.

In this study, the single-agent approach was chosen, and this choice is supported by several considerations:

- Simplicity and Manageability: A single-agent framework simplifies the problem structure. Managing one agent to control all machines reduces the complexity associated with coordination and communication among multiple agents. This simplicity can lead to more straightforward implementation and debugging processes.

- Global Optimization Perspective: A single agent has a global view of the entire job shop environment, enabling it to optimize the scheduling policy considering all machines simultaneously. This holistic approach can potentially lead to better overall performance compared to a scenario where multiple agents might focus on local optimizations that do not necessarily result in a globally optimal solution.

- Resource Efficiency: Training a single agent generally requires fewer computational resources compared to training multiple agents. The learning process for a single agent can be more efficient in terms of both time and computational power, as it avoids the overhead associated with multi-agent interactions.

- Scalability: While multi-agent systems can suffer from scalability issues as the number of agents increases, a single-agent system can be more easily scaled to handle larger job shop environments by enhancing the agent's capacity and learning algorithms without the need to manage inter-agent communications.

Given these reasons, the single-agent RL approach provides a balanced trade-off between implementation complexity and the ability to achieve a globally optimized scheduling policy, making it a suitable choice for tackling the Job Shop Scheduling Problem in this study.

8.2 Theorical aspects of the algorithm

8.2.1 State representation

State representation is a crucial component in the application of Reinforcement Learning to the Job Shop Scheduling Problem. It refers to the way in which the status of the job shop environment is encoded and presented to the RL agent. An effective state representation captures all relevant information about the environment, enabling the agent to make informed decisions that lead to optimal scheduling policies.

The state representation serves several essential purposes:

- Decision Making: It provides the RL agent with the necessary information to decide which job to schedule next for each machine.
- Learning: It allows the agent to learn patterns and dependencies within the job shop environment, improving its ability to predict the outcomes of its actions.
- Environment Interaction: It facilitates the interaction between the agent and the environment, ensuring that the agent’s decisions are based on the current status of the job shop.

In the context of this study, the state is represented as a matrix where each row corresponds to a machine in the job shop, and each column corresponds to a specific feature related to the machine's status. The entries in the matrix provide continuous values rather than binary indicators, offering a more nuanced depiction of the job shop environment.

Specifically, the state matrix includes the following features for each machine:

- Leftover Time on Current Operation: This represents the remaining time required for the machine to complete its current operation.
- Current Operation Percentage: This indicates the percentage of the current operation that has been completed.
- Total Leftover Time: This captures the total remaining time for all operations yet to be completed on the machine.
- Idle Time: This measures the amount of time the machine has been idle.
- Time to Next Operation: This denotes the time until the machine starts its next operation.

The state representation is dynamic and can be updated in real-time to reflect changes in the job shop environment as jobs are processed and new jobs arrive.
The choice of this continuous, feature-based state representation is justified by several considerations:

- Richness and Detail: Unlike a binary matrix, this representation offers a richer and more detailed depiction of the job shop status. Each feature provides specific and quantifiable information that can lead to more informed decision-making by the RL agent.

- Dynamic Adaptability: The continuous nature of the state matrix allows it to be dynamically updated to reflect real-time changes. This is particularly important in environments where the number of jobs is not known a priori, as it ensures that the state representation remains accurate and relevant.

- Efficiency: The continuous feature-based matrix is compact and efficient. This reduces the computational burden on the RL agent when processing state information, enabling faster decision-making and learning.

- Scalability: The matrix representation can be easily scaled to accommodate different numbers of machines and jobs. This flexibility ensures that the approach can be applied to job shops of various sizes without requiring significant changes to the underlying representation.

- Focus on Operational Metrics: By focusing on key operational metrics such as leftover time and idle time, this representation highlights the critical aspects of the job shop environment that directly impact scheduling decisions. This helps the agent to prioritize actions that optimize the flow of jobs through the system.

8.2.2 Action space representation

Action representation is a fundamental aspect of applying Reinforcement Learning to the Job Shop Scheduling Problem. It defines the set of actions available to the RL agent, which, in turn, determines how the agent can interact with the environment to influence the job scheduling process.

The action representation serves several critical functions:

- Decision Making: It provides the agent with a clear set of options for influencing the job scheduling, directly impacting the overall performance and efficiency of the job shop.

- Learning: A well-defined action space allows the agent to explore different scheduling strategies and learn which actions lead to the best outcomes.

- Policy Optimization: The chosen actions are essential for the agent to develop an optimal policy that can handle various job shop scenarios and constraints.

Two strategies are generally used when it comes to defining the action space:

- considering all possible actions available to the agent.
  While considering all possible actions in a deterministic context is usually the best choice, allowing for a higher degree of control from the agent, in the dynamic scenario might entail a constraint in terms of number jobs that the system can handle.

- Considering specific strategies corresponding to the dispatching rules of the priority dispatching rule methods: SPT, LPT, FIFO, LIFO and let the agent choose among those strategies instead of the actual actions.

In this study, actions are defined as scheduling strategies that the agent can choose from. The selected actions have been selected after a profound study of several job shop scheduling problem instances.

- LTPT – Longest total processing time.

  Choosing the job with the longest total processing time remaining across all machines: This strategy prioritizes jobs that require the most time to complete according to its operations duration, ensuring that these more complex jobs are scheduled earlier, potentially reducing their impact on the overall job shop flow.

- HRS – Highest remaining steps.

  Choosing the job with the highest number of remaining steps: This strategy focuses on jobs that have the most steps left in their processing sequence.
  A step is an elaboration phase on a specific machine. By addressing these jobs first, the agent can help to reduce bottlenecks that may arise from jobs that are still far from completion.

- NULL – null action.

  Choosing the null action: This action allows the agent to opt for no immediate scheduling decision, which can be useful in scenarios where delaying a decision might lead to a more optimal scheduling arrangement in subsequent steps.

These actions were chosen based on their ability to address different scheduling challenges and their potential to improve the overall efficiency of the job shop.

8.2.3 Action masking

To enhance the efficiency of the agent's decision-making process, the solution implements an action masking function. Action masking reduces the search space by preventing the agent from selecting illegal or non-optimal actions at each time step. This approach ensures that the agent's choices are always meaningful and contextually appropriate, which improves the learning process and accelerates convergence to an optimal scheduling policy. [17]

Action masking, in this study serves two primary purposes: reducing the search space and preventing non-optimal actions. By eliminating illegal actions, the agent's decision-making process becomes more focused and efficient. This reduction in complexity allows the agent to explore relevant actions more thoroughly. Additionally, action masking helps avoid actions that would not contribute to an optimal schedule, such as choosing to stand by when immediate action is required.

In the implementation, it is ensured that the null action (where the agent chooses to stand by) is only allowed if at least one machine is currently processing a job. This prevents the agent from wasting opportunities to allocate jobs when machines are idle. For instance, if all machines are idle, the agent is forced to select an action that schedules a job, thereby minimizing idle time and improving machine utilization.

When there is only one job in the queue, the environment forces the agent to select the first strategy, which is to choose the job with the longest remaining processing time across all machines. In this scenario, other strategies are redundant and do not provide any additional benefit. For example, if Job A is the only job waiting to be scheduled, the agent automatically applies the first strategy to allocate Job A, ensuring that decision-making is streamlined and relevant.

Incorporating an action masking function significantly enhances the performance of the RL agent in the job shop scheduling environment. By preventing illegal and non-optimal actions, we ensure that the agent's decision-making process is both efficient and effective, ultimately leading to better scheduling outcomes and improved operational efficiency.

Figure 12 and 13 represents the different learning curve in the two cases of PPO with and without action masking. It is noticeable how the learning curve of the Maskable PPO algorithm is steeper with respect to the PPO algorithm, representing a quicker learning phase.

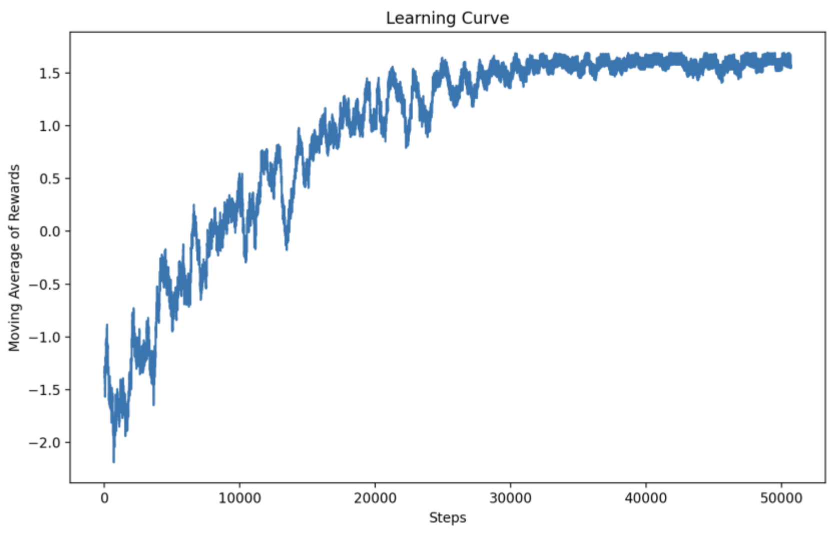


Figure 12: PPO learning curve

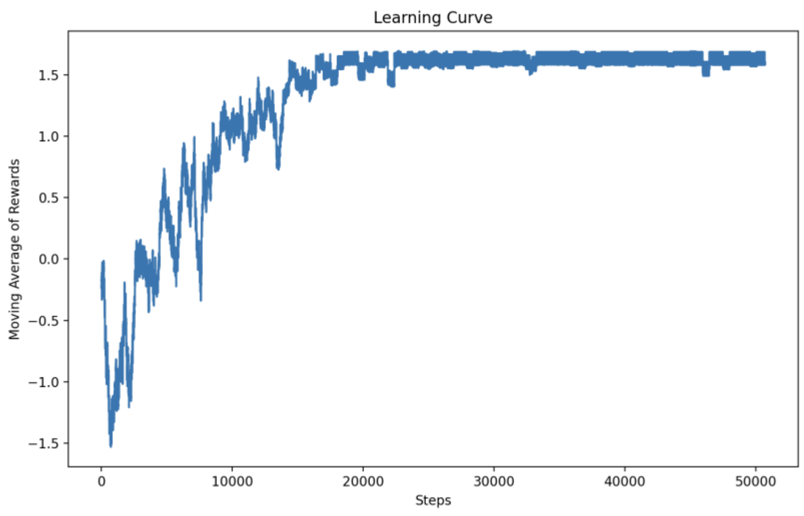


Figure 13: Maskable PPO learning curve

8.2.4 Reward function

Designing a reward function that provides feedback to the agent based on the quality of the schedules generated is one of the most important steps in the definition of a RL algorithm. The reward function should incentivize the agent to generate schedules that maximize the objective function.
An ill-designed reward function can lead to slow learning, suboptimal policies, or even convergence to incorrect behaviors.

A well-designed reward function should:

- Encourage Desired Behaviors: It should incentivize the agent to take actions that lead to optimal scheduling.
- Provide Timely Feedback: It should offer frequent and informative feedback to help the agent quickly learn the consequences of its actions.
- Balance Complexity and Clarity: It should be complex enough to capture important aspects of the problem but clear enough for the agent to interpret and learn from effectively.

The reward strategy explored aims at minimizing idle times (holes) on machines, thereby optimizing machine utilization.

Reward Function:

$$R(s,a) = p_{aj} - \sum_{m \in M} empty_m(s,s')$$

- p_{aj}: Processing time of the scheduled operation.
- $empty_m(s,s')$: Idle time on machine m during the transition from state s to state s'.

Advantages:

- Dense Rewards: This strategy provides frequent feedback after each action, helping the agent to understand the immediate impact of its decisions.
- Maximizes Utilization: By focusing on minimizing idle times, it encourages better machine utilization, closely related to minimizing the makespan.

8.2.5 Environment dynamics and benefit of event-based control

In this study, the job shop environment operates dynamically, with agent-environment interactions triggered by specific events—namely, the availability of one or more machines. Unlike traditional Reinforcement Learning frameworks that rely on fixed time intervals or a predefined number of steps for updates, our approach leverages event-based control, where the agent's internal clock advances only when a new machine becomes available. This method offers several significant advantages:

- Timeliness: By prompting the agent to act only when machine availability changes, the agent's scheduling decisions are made in real-time based on the current state of the job shop.
- Resource Efficiency: Event-based updates focus the agent’s computational effort on significant events, avoiding unnecessary updates that occur in fixed-interval systems.

8.3 Practical implementation

8.3.1 Environment

To train, test and evaluate the proposed algorithm, an environment tailored for the deterministic and dynamic Job Shop Scheduling problem, utilizing the Python programming language has been created.
The environment is designed to facilitate the comparative analysis of multiple scheduling algorithms under both static and dynamic conditions.
Some of the most important algorithms functions are reported in the Appendix.

The core architecture of the environment encompasses several key modules, including a job generator, a scheduler, and a performance evaluator. The job generator simulates various job shop scenarios, incorporating both predefined and stochastic job arrival patterns to reflect the dynamic nature of real-world manufacturing environments. The scheduler module integrates multiple algorithms, allowing for seamless switching and comparative analysis. These algorithms include classical methods such as dispatching rules.
The environment also supports logging and visualization capabilities, enabling detailed analysis of the scheduling process and outcomes.

The implementation in Python provides several advantages, including ease of integration with existing scientific libraries and tools, as well as the flexibility to extend the environment with additional features and algorithms.

Overall, this environment represents a robust and flexible tool for the systematic testing and evaluation of scheduling algorithms, contributing to the advancement of knowledge in the field of Job Shop Scheduling.

8.3.2 Gymnasium and custom environment creation

Gymnasium is an open-source Python library that provides a standard API for reinforcement learning environments. It is a fork and successor of the popular OpenAI Gym library, maintained by the Farama Foundation [81]. Gymnasium offers a wide range of pre-built environments and tools for developing new ones, making it an essential toolkit for reinforcement learning research and development.

The foundational concept in Gymnasium is the environment, which encapsulates the dynamics of the problem space in which an RL agent operates. This environment defines three critical components:

- the state space,
- the action space,
- and the transition dynamics, including reward functions.

In the context of the Job Shop Scheduling Problem, these components are carefully designed to represent the complexities of industrial scheduling scenarios.

In the context of reinforcement learning environments, the Gymnasium library provides a robust framework for defining both observation and action spaces, crucial for agent interaction with the environment. The observation space encompasses the information available to the agent at each decision point, while the action space delineates the possible actions an agent can take. Gymnasium offers a rich taxonomy of space types to define these domains, each tailored to different types of data and interaction paradigms.

Mian types of gymnasium spaces include:

- Discrete Space.

  The Discrete space is used for scenarios where the action or observation can be represented by a finite set of values. For instance, in a simple game with a fixed number of moves, each move can be an element in a Discrete space.

- Box Space.

  The Box space is utilized for continuous spaces and is defined by lower and upper bounds. It is suitable for environments where the agent's observations or actions are multidimensional and can take any value within a specified range, such as the position and velocity of an object.

- MultiDiscrete Space.

  The MultiDiscrete space extends the Discrete space by allowing a fixed number of discrete actions across multiple dimensions. This is particularly useful in environments where multiple, independent discrete actions are required simultaneously.

- MultiBinary Space.

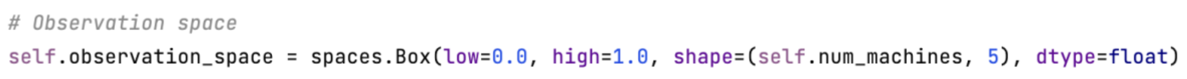
  The MultiBinary space represents observations or actions as binary vectors. It is appropriate for scenarios where each dimension can either be 0 or 1, such as on/off states or presence/absence indicators.

- Dict Space.
  The Dict space enables the combination of multiple Gymnasium spaces into a single composite space. This is beneficial for environments requiring structured and heterogeneous observations or actions, encapsulating different types of data in a unified format.

In the Job Shop Scheduling Problem (JSSP) implementation presented in this thesis, the observation space is modeled using a Dict space. This composite structure allows the integration of multiple subspaces to represent complex and varied data required for decision-making. Specifically, the Dict space combines MultiBinary spaces to capture the status of queues and the availability of machines.

- Queue Status: A MultiBinary space is used to indicate the presence or absence of jobs in various queues. Each bit in the binary vector represents whether a specific job is waiting in the queue.
- Machine Availability: Another MultiBinary space is employed to represent the operational status of machines. Each bit indicates whether a particular machine is available or occupied.

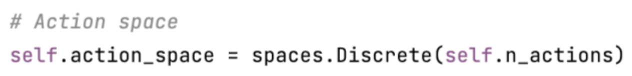
This approach enables a detailed and organized representation of the environment, allowing the agent to receive comprehensive and structured observations.



```
# Observation space
self.observation_space = spaces.Box(low=0.0, high=1.0, shape=(self.num_machines, 5), dtype=float)
```

Figure 14: Observation space definition

The action space, which delineates the set of possible decisions available to the agent, is represented using a Discrete space in this implementation:



```
# Action space
self.action_space = spaces.Discrete(self.n_actions)
```

Figure 15: Action space definition

This choice reflects the discrete nature of job selection strategies in the JSSP context, where each action corresponds to a specific scheduling heuristic or rule.

In the design and implementation of reinforcement learning (RL) environments, two pivotal functions provided by the Gymnasium framework are the step and reset functions. These functions are fundamental to the interaction between the agent and the environment, enabling the agent to learn and adapt through iterative cycles of action and feedback.

The step function is central to the dynamic progression of the environment. It advances the environment by one-time step-in response to an action taken by the agent. This function performs several critical tasks:

- Action Processing: The function receives an action from the agent, which is an element of the defined action space.

- State Transition: Based on the given action, the environment transitions to a new state. This involves updating the environment's internal variables and configurations to reflect the consequences of the action.

- Reward Calculation: The function computes a reward signal that quantifies the immediate benefit or cost of the action taken. This reward is pivotal for the RL agent’s learning process, as it provides the feedback necessary to evaluate the desirability of actions.

- Termination Check: The function determines whether the current episode has reached a terminal state. This could be due to achieving a goal, reaching a maximum number of steps, or encountering an error state.

- Information Provision: Additional diagnostic information may also be returned to aid in debugging and understanding the environment’s behavior.

The step function typically returns a tuple containing the new state, the reward, a Boolean indicating whether the episode has ended, and an optional dictionary of additional information.

The reset function is equally crucial in the context of RL training, as it reinstates the environment to its initial configuration at the start of each new episode. This function ensures consistency and reproducibility in the agent's learning process by providing a stable starting point for each episode. The key aspects of the reset function include:

- Environment Initialization: The function reinitializes the environment's state to a predefined starting configuration, which is often randomly chosen from a set of possible initial states to ensure diverse learning experiences.

- State Return: After resetting, the function returns the initial state of the environment, which becomes the starting point for the agent's next series of actions.

The consistent reinitialization provided by the reset function is fundamental for episodic learning, allowing the agent to repeatedly explore and learn from different scenarios starting from a known state.

Together, the step and reset functions form the core interaction loop of RL agents within Gymnasium environments. During training, the agent repeatedly performs the following cycle:

- Reset: Begin a new episode by calling the reset function to obtain the initial state.
- Act: Select an action based on the current policy.
- Step: Execute the action using the step function to transition to a new state, receive a reward, and check for episode termination.
- Learn: Update the agent's policy based on the received reward and new state.
- Repeat: Continue acting and stepping until the episode terminates, then reset the environment to start a new episode.

This interaction loop enables the agent to explore the environment, learn optimal policies through trial and error, and improve performance over time.

8.3.3 Proximal policy optimization

In this research, it is leveraged the implementation of PPO provided by Stable Baselines3 [80], a library recognized for its robust implementations of reinforcement learning algorithms. Specifically, the Maskable Actor Critic Policy, which is a variant of the actor-critic architecture capable of handling environments with masked actions, has been employed.

PPO's approach to optimization strikes a balance between exploration and exploitation, making it suitable for tasks requiring both robust learning and efficient policy updates. The benefit of using Stable Baselines3 is a straight-forward implementation process and the assurance of working with a reliable framework designed to support standardized reinforcement learning experiments.

8.3.4 Hyper-parameter search

Fine-tuning is a crucial process in the field of machine learning, particularly in reinforcement learning (RL) and deep learning. It involves the optimization of hyperparameters to improve the performance of a model on a specific task.

Fine-tuning plays a pivotal role in enhancing the efficacy of machine learning models. It allows for the adjustment of model parameters to better fit the specific characteristics of a given task or dataset. In reinforcement learning, fine-tuning is particularly important due to the sensitivity of RL algorithms to hyperparameter settings [82].

The importance of fine-tuning in RL can be attributed to several factors:

1. Task Specificity: Different tasks require different hyperparameter configurations for optimal performance. Fine-tuning allows the model to adapt to the specific nuances of the job shop scheduling problem [83].

2. Sample Efficiency: Proper hyperparameter tuning can significantly improve sample efficiency, allowing the model to learn more effectively from a limited number of interactions with the environment [84].

3. Stability: Fine-tuning can enhance the stability of the learning process, reducing the variance in performance across different runs.

4. Performance Optimization: By systematically exploring the hyperparameter space, fine-tuning aims to find the configuration that yields the best performance on the given task.

The most important Proximal Policy Optimization (PPO) algorithm parameters include:

- n_steps: This parameter determines the number of steps collected per environment before updating the policy. It's crucial for balancing between sample efficiency and computational cost.
    - Larger values can lead to more stable updates but may slow down learning.
    - Smaller values allow for more frequent updates but might increase variance.
    - Typical range: 1024 to 8192 [85]

- gamma (Discount factor): Gamma represents the discount factor for future rewards. It determines the importance of future rewards compared to immediate rewards.
    - Values close to 1 prioritize long-term rewards.
    - Lower values focus more on immediate rewards.
    - Typical range: 0.9 to 0.999 [82]

- learning_rate: The learning rate controls the step size at each iteration while moving toward a minimum of the loss function.
    - Too high: May cause unstable training or divergence.
    - Too low: May result in slow convergence.
    - Often uses adaptive methods like Adam optimizer [86]
    - Typical range: 1e-5 to 1e-3

- clip_range: This is a key parameter in PPO, used to clip the policy ratio to prevent excessively large policy updates.
    - Helps maintain proximity between the old and new policies.

    - Typical value: 0.2 (can be annealed over time)
    - Range often explored: 0.1 to 0.3 [79]

- gae_lambda: Used in Generalized Advantage Estimation (GAE), this parameter controls the trade-off between bias and variance in advantage estimation.
    - λ = 1 gives high variance, unbiased estimates.
    - λ = 0 gives low variance, but biased estimates.
    - Typical range: 0.9 to 1.0 [79]

Other important parameters not directly tuned in this code but worth mentioning:

- ent_coef (Entropy coefficient): Encourages exploration by adding an entropy bonus to the objective.
    - Higher values promote more exploration.
    - Typical range: 0.0 to 0.01

- vf_coef (Value function coefficient): Determines the importance of the value function loss in the overall loss function.
    - Balances between policy and value function optimization.
    - Typical range: 0.5 to 1.0

- max_grad_norm: Used for gradient clipping to prevent exploding gradients.
    - Helps maintain training stability.
    - Typical value: 0.5

The interplay between these parameters significantly influences the performance of the PPO algorithm. It's important to note that the optimal values for these parameters can vary greatly depending on the specific task and environment. This is why hyperparameter tuning has been implemented to achieve optimal performance.

The code employs Optuna, a hyperparameter optimization framework, to conduct the fine-tuning process [87].

Key aspects of the method:

- Hyperparameter Space Definition: The code defines a search space for five key hyperparameters of the PPO algorithm:

    - n_steps: Number of steps per update (2048 to 8192)

    - gamma: Discount factor (0.8 to 0.9999)

    - learning_rate: Learning rate (1e-5 to 1e-3)

    - clip_range: PPO clip range (0.1 to 0.4)

    - gae_lambda: GAE lambda parameter (0.8 to 1.0)

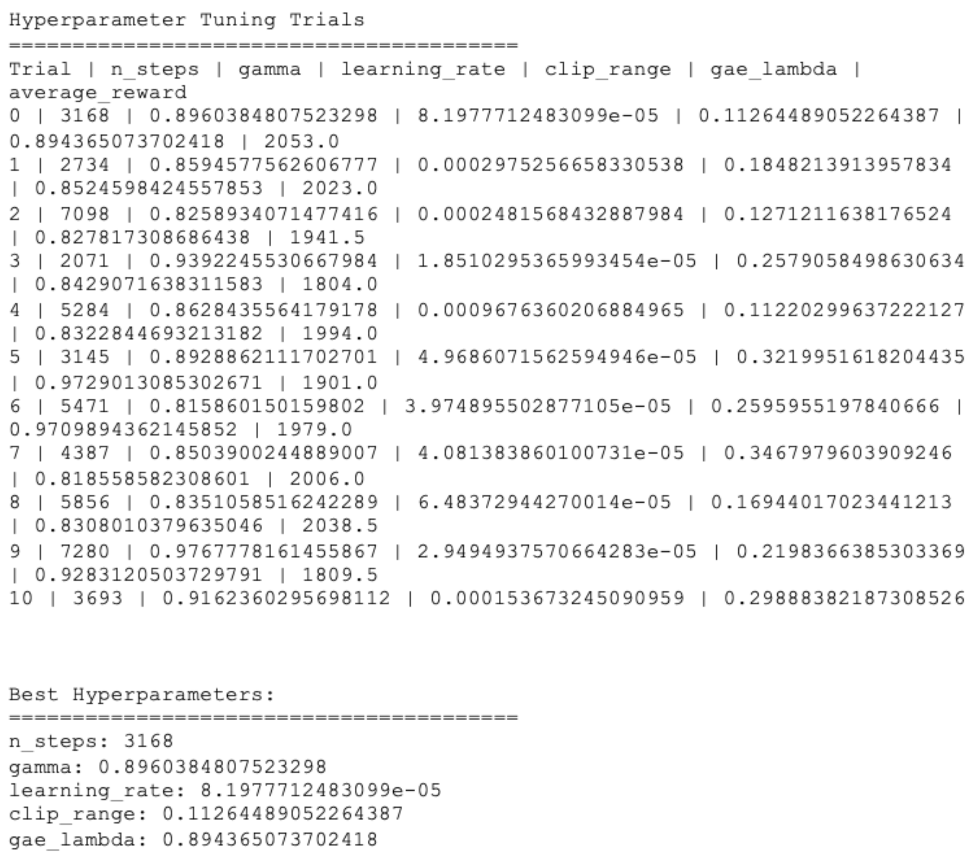
- Objective Function: An objective function is defined that trains a PPO model with the given hyperparameters and evaluates its performance. The evaluation metric combines both the average reward over 10 episodes and a speed score based on the moving average of rewards during training.

- Optimization Process: Optuna conducts 50 trials, each with a different hyperparameter configuration suggested by the TPE algorithm. For each trial, the PPO model is trained for 70,000 timesteps.

- Best Configuration Selection: The hyperparameter configuration that yields the highest combined score (average reward + speed score) is selected as the best configuration.

- Retraining: After the optimization process, the model is retrained using the best hyperparameters found.

Below are reported few lines of the tuning log showing the fine-tuning process and the best hyperparameters reached.



```
Hyperparameter Tuning Trials
========================================
Trial | n_steps | gamma | learning_rate | clip_range | gae_lambda |
average_reward
0 | 3168 | 0.8960384807523298 | 8.1977712483099e-05 | 0.11264489052264387 |
0.894365073702418 | 2053.0
1 | 2734 | 0.8594577562606777 | 0.0002975256658330538 | 0.1848213913957834
| 0.8524598424557853 | 2023.0
2 | 7098 | 0.8258934071477416 | 0.0002481568432887984 | 0.1271211638176524
| 0.827817308686438 | 1941.5
3 | 2071 | 0.9392245530667984 | 1.8510295365993454e-05 | 0.2579058498630634
| 0.8429071638311583 | 1804.0
4 | 5284 | 0.8628435564179178 | 0.0009676360206884965 | 0.11220299637222127
| 0.8322844693213182 | 1994.0
5 | 3145 | 0.8928862111702701 | 4.9686071562594946e-05 | 0.3219951618204435
| 0.9729013085302671 | 1901.0
6 | 5471 | 0.815860150159802 | 3.974895502877105e-05 | 0.2595955197840666 |
0.9709894362145852 | 1979.0
7 | 4387 | 0.8503900244889007 | 4.081383860100731e-05 | 0.3467979603909246
| 0.818558582308601 | 2006.0
8 | 5856 | 0.8351058516242289 | 6.48372944270014e-05 | 0.16944017023441213
| 0.8308010379635046 | 2038.5
9 | 7280 | 0.9767778161455867 | 2.9494937570664283e-05 | 0.2198366385303369
| 0.9283120503729791 | 1809.5
10 | 3693 | 0.9162360295698112 | 0.000153673245090959 | 0.29888382187308526


Best Hyperparameters:
========================================
n_steps: 3168
gamma: 0.8960384807523298
learning_rate: 8.1977712483099e-05
clip_range: 0.11264489052264387
gae_lambda: 0.894365073702418
```

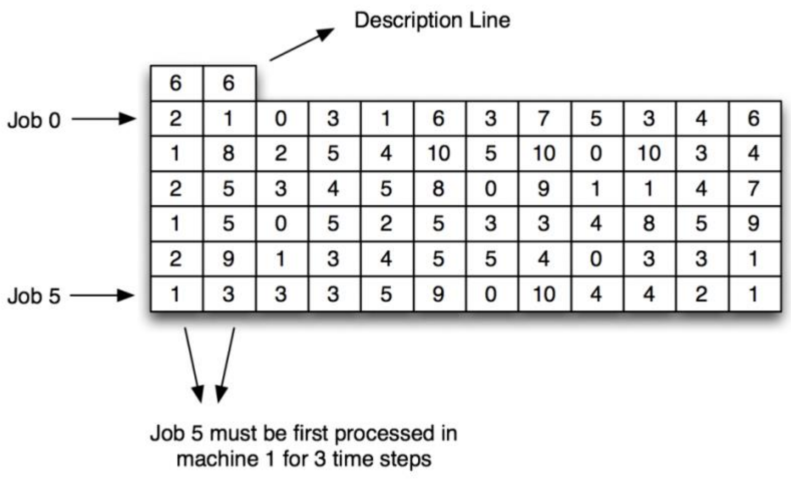
Figure 16: Fine-tuning process

9 Reinforcement learning in deterministic environments

9.1 Benchmark job shop scheduling problems

The OR library [88] collects test data for a variety of Operational Research problems including the job shop scheduling problem.
The benchmark problems contained in the library are extensively used in the literature to test resolution methods and algorithms.

The standard notation of the benchmark problems is exemplified below:



6	6										
2	1	0	3	1	6	3	7	5	3	4	6
1	8	2	5	4	10	5	10	0	10	3	4
2	5	3	4	5	8	0	9	1	1	4	7
1	5	0	5	2	5	3	3	4	8	5	9
2	9	1	3	4	5	5	4	0	3	3	1
1	3	3	3	5	9	0	10	4	4	2	1

Figure 17: Example of a JSSP instance (ft06) from the OR-Library [88]

The first line specifies the number of job and machines of the problem instance.
Below a matrix reports the processing order and times of each job. Particularly, every row corresponds to a job and contains the ordered sequence of operations associated with the job. Each operation is expressed as a pair of values composed of the machine identification number (machine id starts from number 0) and the processing time of the operation.

The optimal solution for each benchmark problem is known. For instance, the optimal solution of the benchmark problem above reveals a make span of 55 time units.

9.2 Results

The proposed algorithm has been tested it against a series of JSSP established benchmark problems collected in the OR library and its performance has been compared with traditional methods including priority dispatching rules (FIFO, LPT, SPT), and genetic algorithm literature results[19].

This approach is crucial for several reasons. Firstly, benchmark problems provide a standardized platform to assess the effectiveness, robustness, and efficiency of the algorithm, ensuring that the results are comparable and reproducible. Furthermore, by comparing the performance of our algorithm with traditional methods, we can objectively determine its relative strengths and weaknesses. This comparison not only highlights areas where our

algorithm excels but also identifies potential limitations, thereby providing a balanced and thorough assessment of its practical utility.
Such rigorous testing and comparison are essential to validate the algorithm's real-world applicability and to demonstrate its potential advantages over existing solutions.

The primary metric for comparison is the makespan, which is the total time required to complete all jobs. Benchmark problems are associated to specific optimal makespan.

Table 1 summarize the results for three benchmark problems demonstrate that the proposed reinforcement learning approach consistently outperforms traditional priority dispatching rules (FIFO, LPT, SPT) and achieves makespan that are comparable to those obtained by the genetic algorithm (GA).

The last three columns refer to the developed solutions including a Q-learning application and the performances of the Proximal policy Optimization method with and without the action masking.

Table 3: Maskable PPO performance in deterministic environment

	OPT	**FIFO**	**LPT**	**SPT**	**GA**	**Q-LEARNING**	**PPO**	**MASKABLE PPO**
FT06 (06X06)	55	65	77	88	55	60	59 (50000 timesteps)	**55** (50000 timesteps)
FT10 (10X10)	930	1184	1295	1074	994	1236	1117 100000 timesteps)	1103 (100000 timesteps)
LA01 (05X10)	666	772	822	751	667	736	712 100000 timesteps)	**666** (100000 timesteps)

10 Reinforcement learning on dynamic environment

10.1 Elements of dynamicity

After identifying the algorithm that performed best on the deterministic Job Shop Scheduling Problems (JSSP), the effectiveness of the solution is further analyzed in a dynamic version of the problem. The dynamicity is introduced thanks to three key elements of variability:

1. Random Job Arrival Time: Jobs do not arrive at the system simultaneously. Instead, each job arrives at a random time, simulating a more realistic production environment where jobs are continuously fed into the system.
2. Variability in Job Processing Time: The processing time for each job is no longer fixed but varies according to a uniform distribution. This introduces uncertainty, requiring the algorithm to adapt to fluctuating processing requirements.
3. Variability in job's processing sequence and times: the jobs entering the system do not belong to a standardized pool as it is for the usual deterministic job shop scheduling problem. They are defined by their specific processing sequence and times.

Testing the algorithm in this dynamic environment provides several advantages:

- Realistic Scenario Simulation: It closely mimics real-world job shop conditions where new jobs arrive unpredictably, and processing times are not always known in advance.
- Robustness Assessment: Evaluating the algorithm's performance under these conditions helps determine its robustness and adaptability to changes and uncertainties.
- Enhanced Learning: The algorithm learns to handle variability in both job arrival and processing times, potentially leading to more flexible and resilient scheduling policies.

By incorporating these dynamic elements into the JSSP, it is possible to understand the algorithm's practical applicability and its potential to improve job shop efficiency in more complex, real-world settings.

10.2 Results

To assess the performance of the proposed algorithm in a dynamic environment, the maskable PPO algorithm has been trained on four different scenarios for a total of 100 000 timesteps.
Following the training phase, 300 episodes were simulated to assess the effectiveness of the learned scheduling policy.
For comparison, 300 episodes were also run using First-In-First-Out (FIFO), Shortest Processing Times (SPT) and Longest Processing Times (LPT) rules, which are one of the most used heuristics in dynamic scheduling scenarios.
The genetic algorithm instead has been tested on only 100 episodes due to an increased running time per episode in comparison to other methods.

The makespan and the total workload has been recorded for each episode under each scheduling strategy.

The workload is defined as the sum of the processing times of each job entered in the system on each machine.

For each scenario and strategy, the registered makespan and workload for each episode have been plotted in a graph and the graph points have been interpolated to depict the the episode durantion trend in relation to an increasing system workload.

The test has been conducted in four different cases.

10.2.1 CASE 1: variability in processing times

The first dynamic environment scenario can be considered as a standardized scenario in which the jobs arriving in the system belongs to a pool of tasks, known to the system.
In the context of standardized production, a factory might produce a limited number of standardized products with well-defined processing sequences. The scheduler stores the processing sequences and times for these standard products as all orders fall within predefined categories.
The environment, however, involves a variability in job processing times which is determined by a uniform distribution.
In this scenario, defining as "$value$" the standard job processing time, the real processing time might vary between $\left[value - \frac{value}{2} \;,\; value + \frac{value}{2}\right]$.

In this scenario, not only the coefficient of the line on the Maskable PPO performance graph is lower compared to other methods, indicating that Maskable PPO achieves a lower makespan per episode in comparison to the other methods, but the graphs show a significantly reduced variability of episodes' duration by using Maskable PPO algorithm instead of other methods.

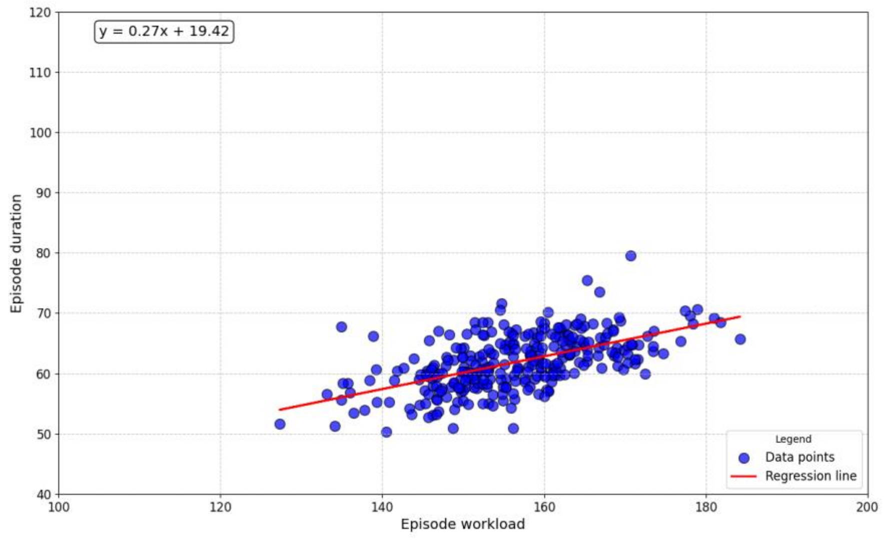


Figure 18: Maskable PPO performance

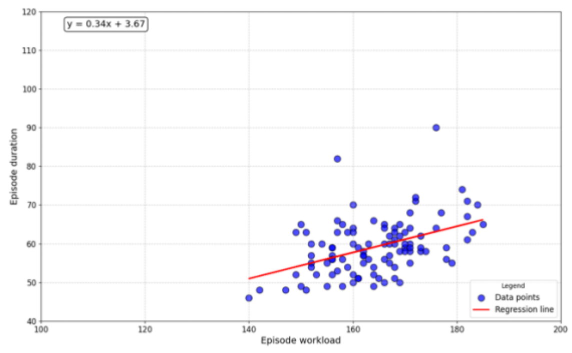


Figure 19: GA performance

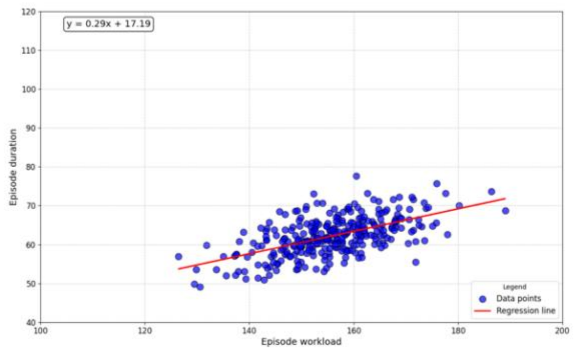


Figure 20: FIFO performance

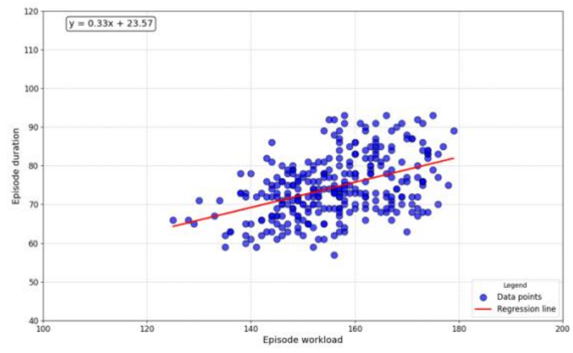


Figure 21: LPT perfomance

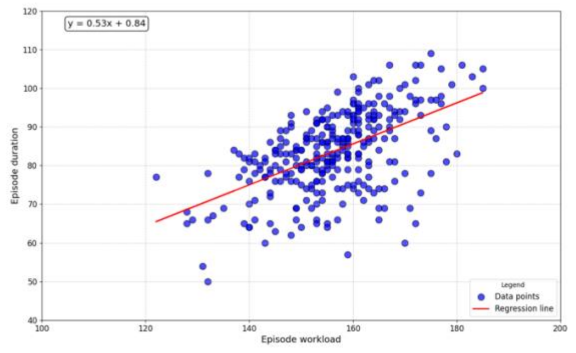


Figure 22: SPT perfomance

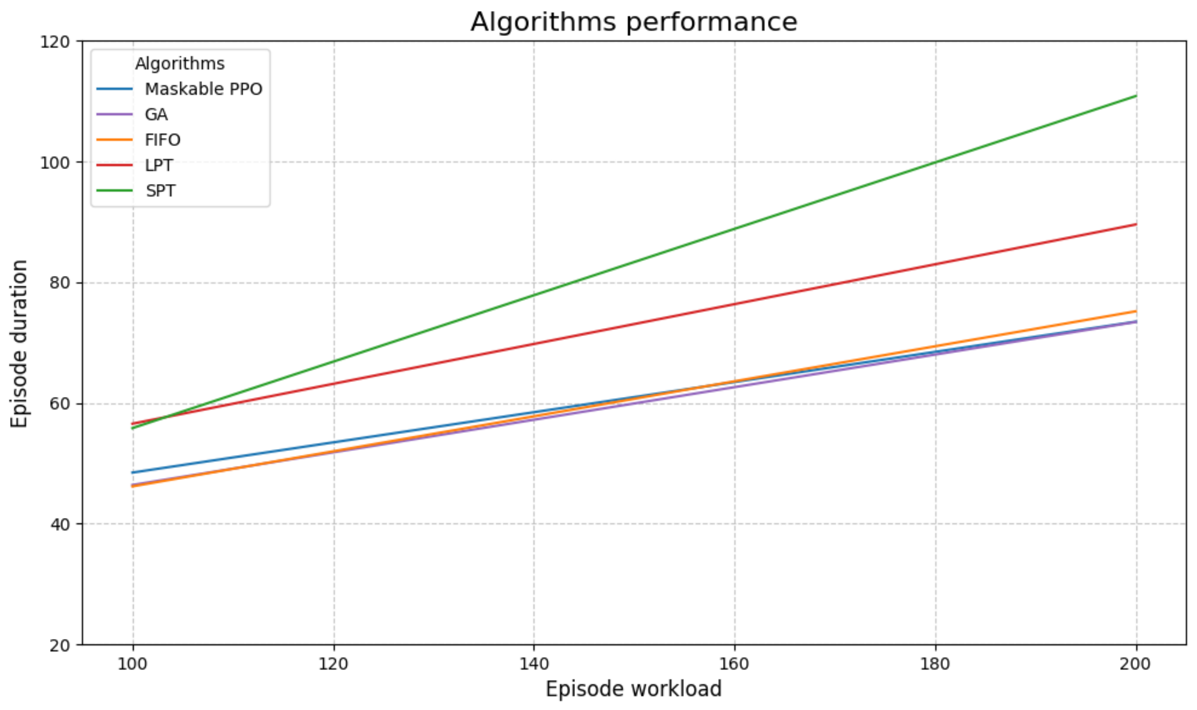


Figure 23: Algorithms' comparison - Case 1

10.2.2 CASE 2: dynamic job arrival

The second dynamic environment scenario is also standardized scenario in which the jobs arriving in the system belongs to a pool of tasks, known to the system.

The key aspect of the case, however, is the variability in job arrival times which are governed by a uniform distribution.
In practical settings, such as in service industries or project management, the variability in job arrival times often follows a uniform distribution, mirroring the case presented. This stochastic nature can impact resource allocation and scheduling strategies, highlighting the relevance of understanding and modeling such distributions in real-world applications.

Notably, the coefficient of the line on the maskable PPO performance graph is lower compared to the other methodologies considered, indicating that Maskable PPO achieves a lower makespan per episode in comparison to the other methods for the same number of total jobs in the system.

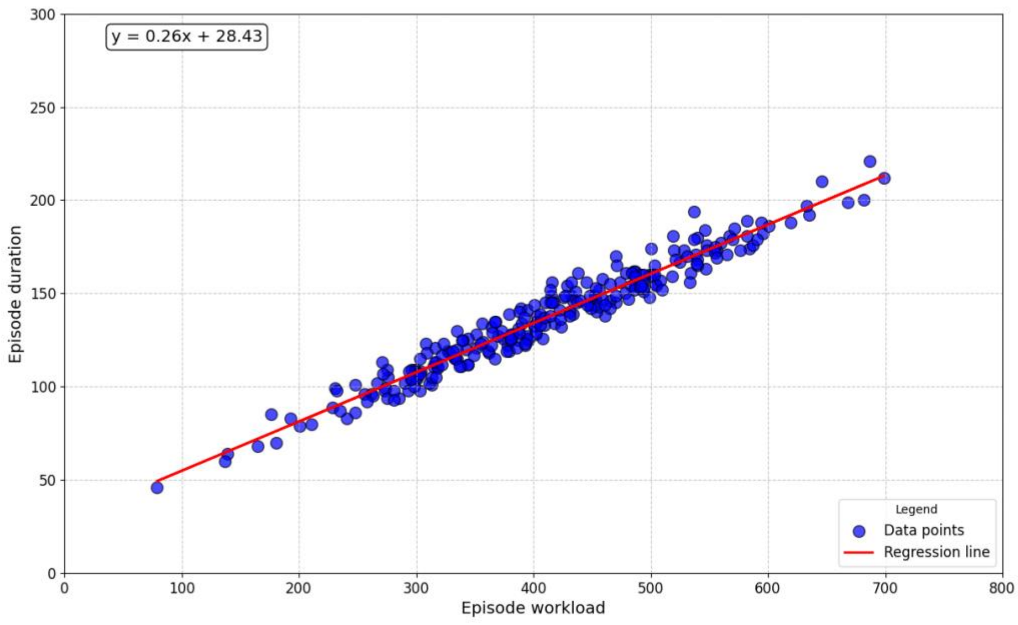


Figure 24: Maskable PPO performance

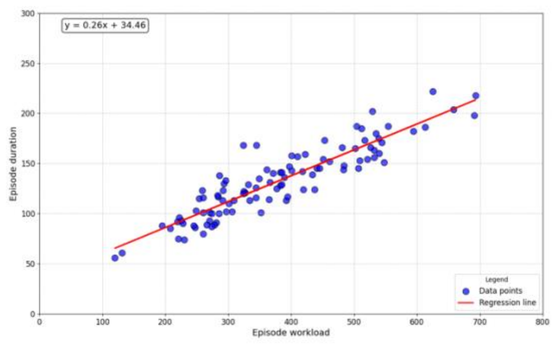


Figure 25: GA performance

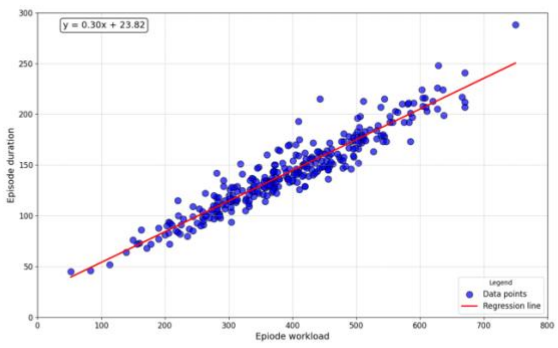


Figure 26: FIFO performance

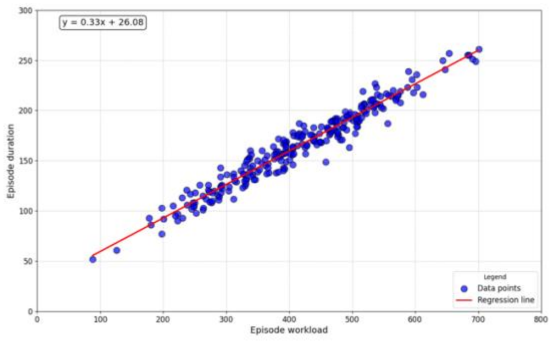


Figure 27: LPT performance

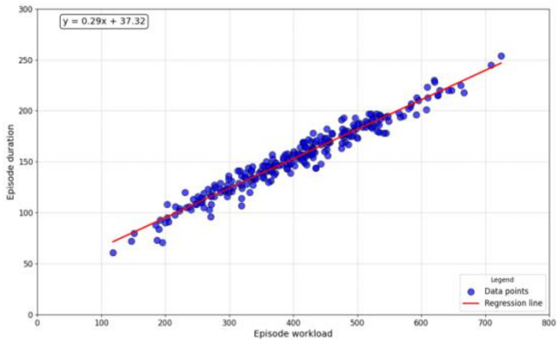


Figure 28: SPT performance

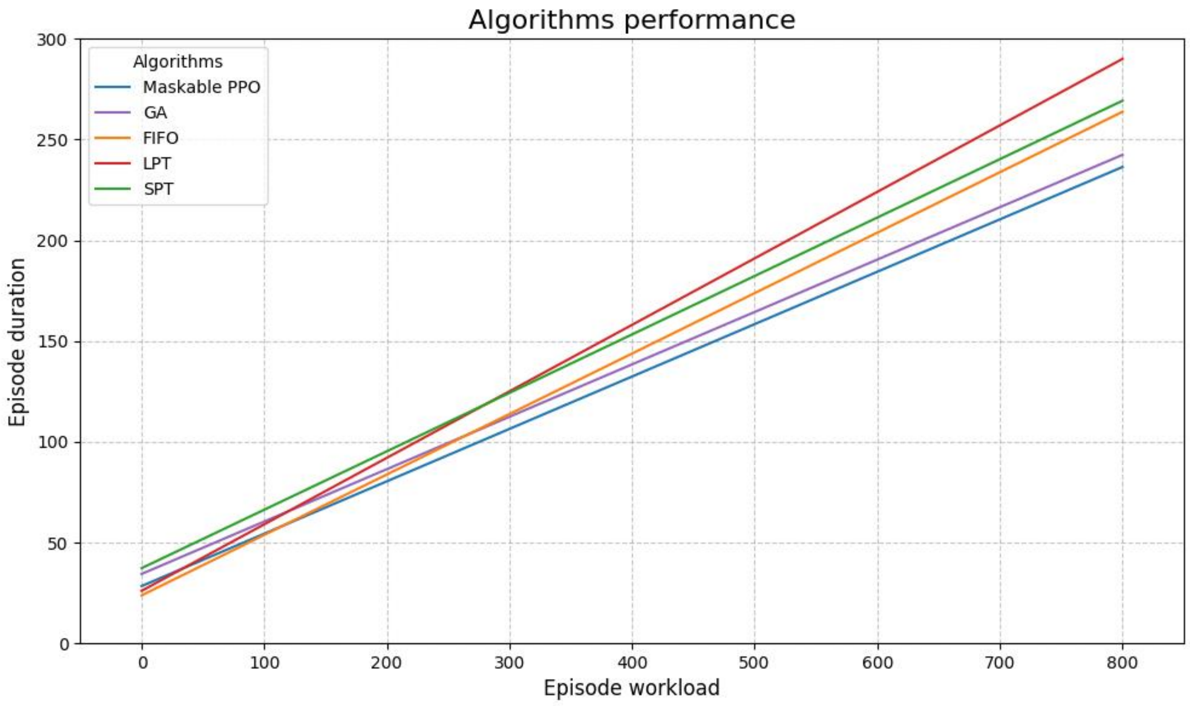


Figure 29: Algorithms comparison - Case 2

10.2.3 CASE 3: dynamic job arrival and variability in processing times

The third dynamic environment scenario constitute a combination of the two cases above involving variability in both job processing times and job arrival.

As for the cases above, the Maskable PPO algorithm continues to the other algorithms highlighting its robust scheduling capabilities in highly dynamic environments.

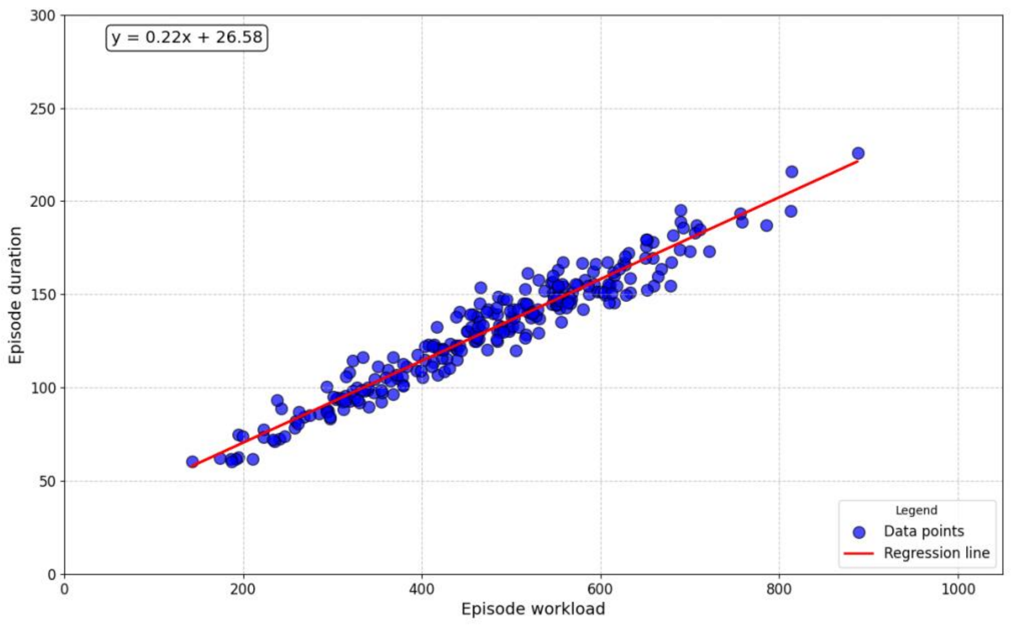


Figure 30: Maskable PPO performance

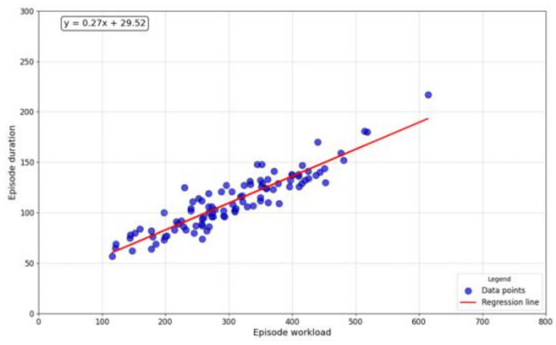


Figure 31: GA performance

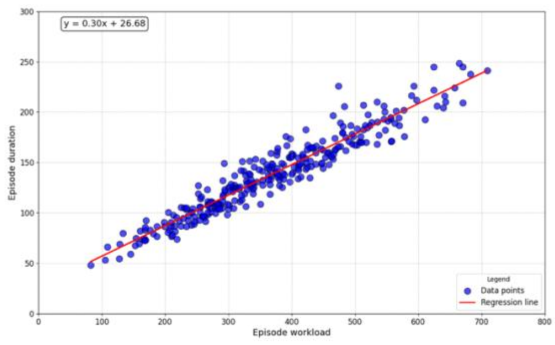


Figure 32: FIFO performance

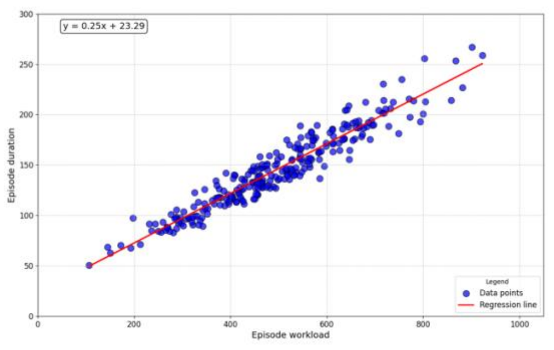


Figure 33: LPT performance

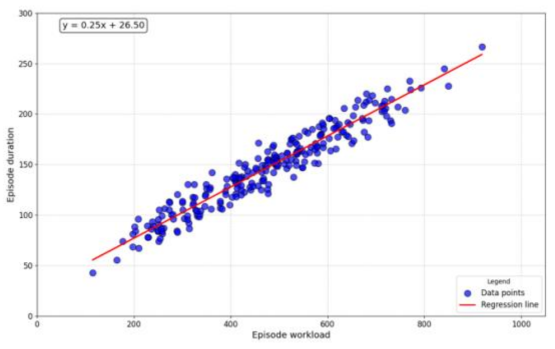


Figure 34: SPT performance

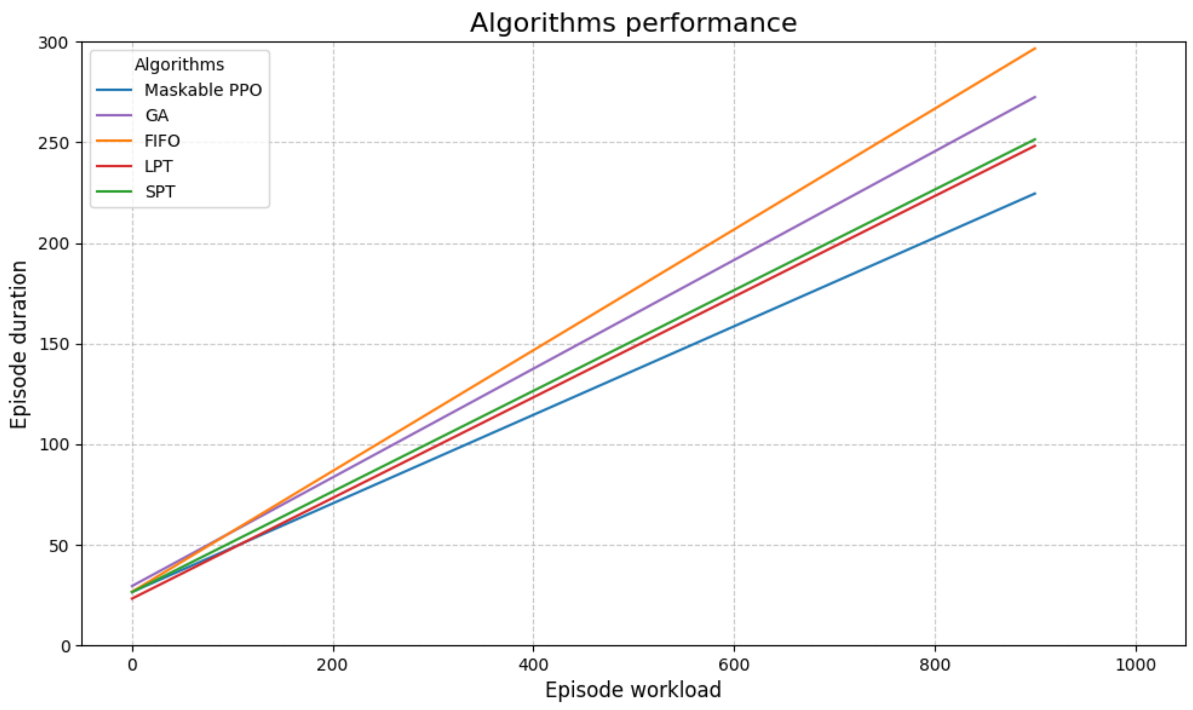


Figure 35: Algorithms' comparison - Case 3

10.2.4 CASE 4: dynamic job arrival and variability in processing times – unknown tasks

The fourth dynamic environment scenario builds upon the preceding one by incorporating variability in both job arrival and job processing times. The key novelty in this case is the introduction of jobs that are unknown to the system. At each time step, a random number of jobs enter the system, each with its own specific processing sequence and times for each machine.

Considering this scenario is important because it reflects the complexity and unpredictability found in many real-world environments. Businesses and organizations often face situations where new tasks and requirements emerge unexpectedly, necessitating a flexible and adaptive scheduling system to maintain efficiency and competitiveness.
To illustrate this, we can examine various contexts where such a scenario plays a critical role. The of standardized production context offers simplicity and efficiency, however, the system's flexibility is limited, meaning it may not quickly adapt to production changes or new products. In contrast, customized production involves customized products based on customer requests for which specifications and processing sequences can vary widely.
The scheduler in this scenario has the capability to accept and manage new, unseen jobs or tasks, quickly integrating new specifications. This approach offers significant flexibility, allowing the system to easily adapt to new orders and specifications, thereby increasing customer satisfaction and market competitiveness. However, this system is more complex to implement and requires continuous updating of information.

In the healthcare sector, hospitals need to efficiently manage resources such as operating rooms, medical equipment, and healthcare personnel to treat patients with varying medical

needs. The scheduler allocates resources by assigning operating rooms, equipment, and personnel based on availability and case priority, optimizes times to reduce patient wait times and improve resource efficiency, and manages emergencies by quickly reallocating resources as needed.
In the transportation sector, logistics companies manage a fleet of vehicles to deliver goods to various destinations with variable timing. The scheduler in this scenario might optimize routes by planning the most efficient delivery routes to minimize travel time and costs, manages the fleet by assigning vehicles and drivers based on availability and delivery requirements, and adapts to changes by adjusting to unforeseen delays, traffic, and new delivery requests.

Figure 6 and 7 depict, for each episode, the total workload of the system and the episode duration.
The points in the graphs have been interpolated to show the episode duration trend in relation to an increasing plant workload.
As for the cases above, the Maskable PPO algorithm continues to outperform FIFO, SPT, and LPT showing a coefficient of the line that is lower compared to other methods and highlighting its robust scheduling capabilities in highly dynamic environments.
Overall, the results clearly indicate that the Maskable PPO algorithm outperforms priority dispatching rules in all tested scenarios, demonstrating its ability to effectively handle the dynamic job shop scheduling problem. By leveraging event-based control and action masking, the Maskable PPO algorithm makes more informed and strategic scheduling decisions, leading to a significant reduction in makespan compared to FIFO, SPT, and LPT rules. These findings high- light the potential of reinforcement learning, in optimizing scheduling policies for dynamic job shop environments. The algorithm’s adaptability to changing conditions and its ability to avoid illegal and non-optimal actions contribute to its superior performance.

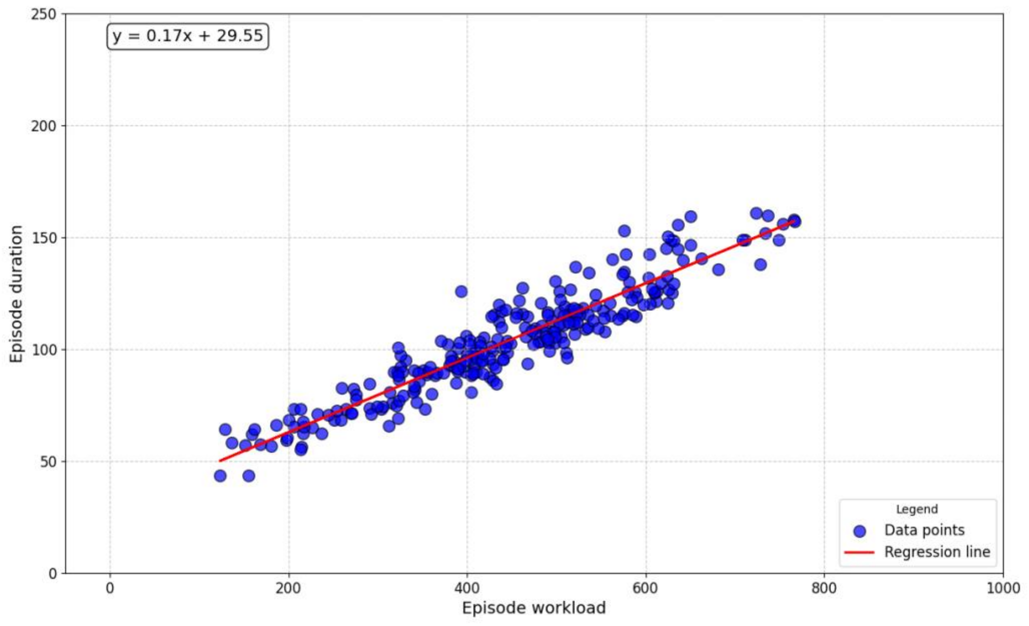


Figure 36: Maskable PPO performance

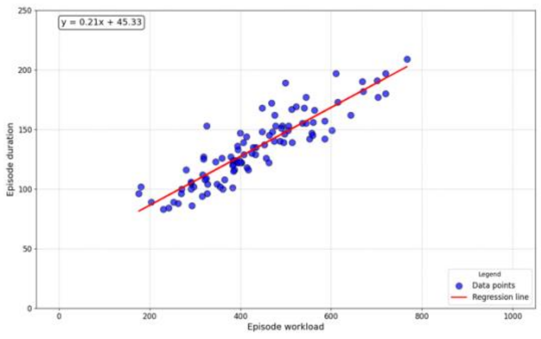


Figure 37: GA performance

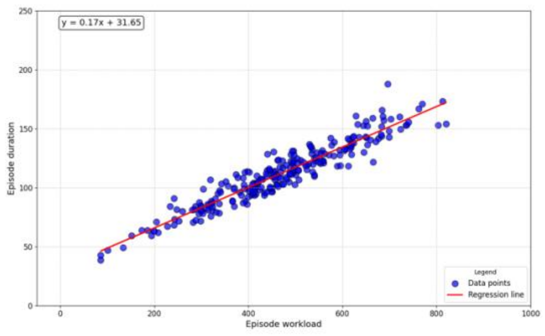


Figure 38: FIFO performance

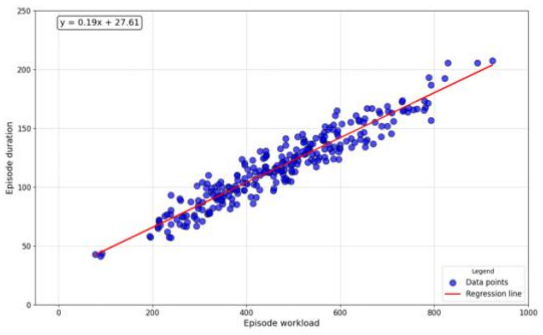


Figure 39: LPT performance

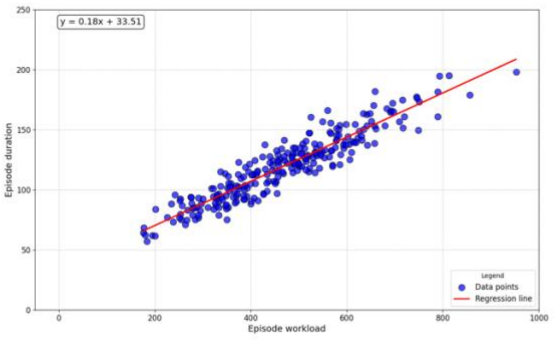


Figure 40: SPT performance

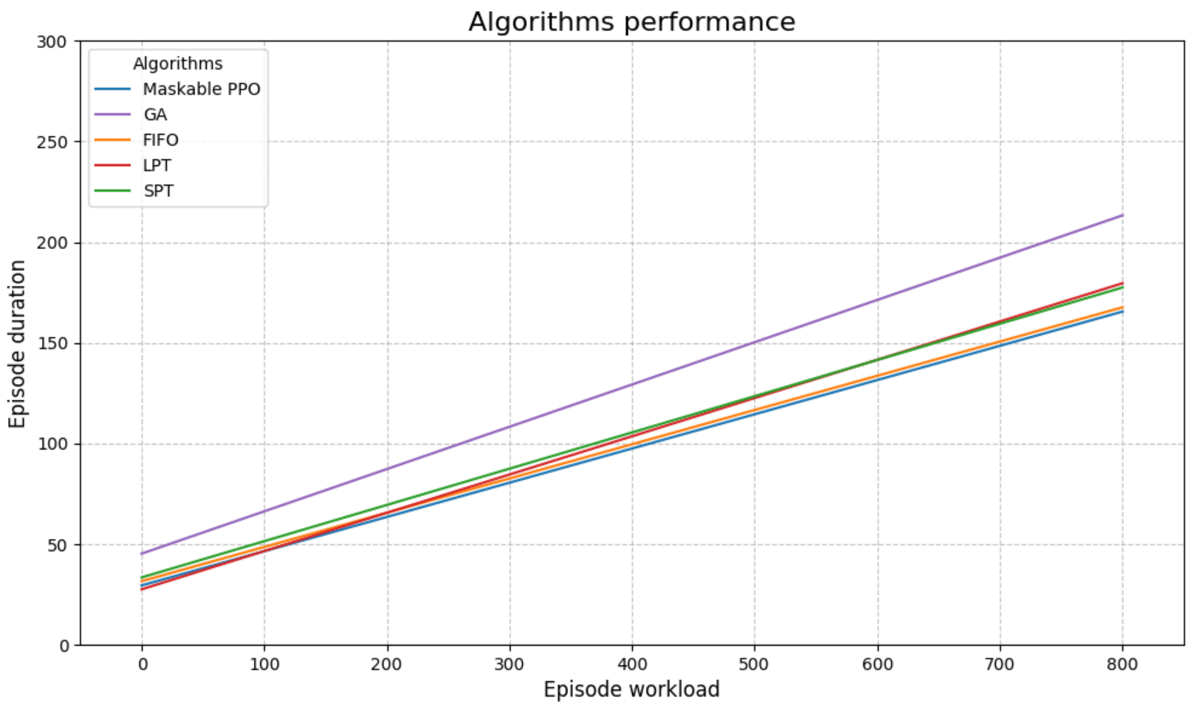


Figure 41: Algorithms' comparison – Case 4

Overall, the results clearly indicate that the Maskable PPO algorithm outperforms priority dispatching rules and the genetic algorithm proposed in all tested scenarios, demonstrating its ability to effectively handle the dynamic job shop scheduling problem.
By leveraging event-based control and action masking, the Maskable PPO algorithm makes more informed and strategic scheduling decisions, leading to a significant reduction in makespan.
These findings highlight the potential of reinforcement learning, in optimizing scheduling policies for dynamic job shop environments and the algorithm’s adaptability to changing conditions.

11 Conclusion

This thesis proposes a single-agent reinforcement learning approach to tackle the Job Shop Scheduling Problem, assessing its performance in both deterministic and dynamic environments.
The proposed method leverages maskable proximal policy optimization to enhance scheduling efficiency.

In deterministic scenarios, the Maskable PPO algorithm consistently achieved optimal or near optimal makespans, demonstrating performance on par with the Genetic Algorithm and surpassing traditional heuristic methods highlighting its capability to effectively manage fixed scheduling tasks.

In dynamic environments, characterized by random job arrivals and variability in processing times, the Maskable PPO algorithm excelled by adapting swiftly to changing conditions and making informed, strategic scheduling decisions. Its adaptability and robust decision-making process resulted in superior performance compared to conventional approaches like priority dispatching rules.

These findings emphasize the significant potential of reinforcement learning, particularly the Maskable PPO algorithm, in addressing and optimizing complex scheduling challenges across various industrial contexts.

12 Future work

Future work on the single-agent reinforcement learning approach for the Job Shop Scheduling Problem will focus on refining the algorithm to further enhance its performance and adaptability.

One area of improvement involves testing various reward functions to better align the reinforcement learning process with specific scheduling objectives, such as minimizing energy consumption or balancing workload across machines.
Additionally, exploring multi-agent reinforcement learning approaches could provide a more robust solution by enabling collaboration and competition among agents, thereby improving overall scheduling efficiency and adaptability in complex manufacturing environments. Another promising direction is the integration of advanced optimization techniques, such as metaheuristic algorithms or hybrid methods that combine reinforcement learning with traditional optimization approaches. This could help in overcoming local optima and achieving better global performance.

To bridge the gap between theoretical optimization and practical implementation, future research will explore the development of digital twin solutions.
By linking the Maskable PPO algorithm to simulation software and integrating it with the physical machines in the manufacturing plant, a real-time scheduling solution can be created. This digital twin approach will enable continuous monitoring and optimization of the

scheduling process, allowing for immediate adjustments based on real-time data from the production floor.
The implementation of a digital twin would involve creating a virtual replica of the manufacturing system, where the reinforcement learning algorithm can be tested and refined in a simulated environment before deployment. This not only ensures the robustness of the scheduling solution but also allows for proactive identification and resolution of potential issues. By continuously updating the digital twin with real-time data from the plant, the algorithm can adapt to changing conditions, such as machine breakdowns or urgent job requests, ensuring optimal scheduling decisions at all times.

In summary, future research will focus on enhancing the reinforcement learning algorithm through improved reward functions, multi-agent approaches, and hybrid optimization techniques. Additionally, the integration of digital twin solutions promises to provide a seamless and adaptive real-time scheduling system, significantly improving operational efficiency and responsiveness in dynamic manufacturing environments.

Bibliography

[1] Schwab, Klaus. The fourth industrial revolution. Crown Currency, 2017.
[2] Brettel, Malte, et al. "How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 Perspective." International journal of information and communication engineering 8.1 (2014): 37-44.
[3] Kagermann, H., Wahlster W., Helbig, J. (2013). Recommendations for implementing the strategicinitiative Industrie 4.0: Final report of the Industrie 4.0 Working Group.
[4] Sharma, Ashwani, and Bikram Jit Singh. "Evolution of industrial revolutions: a review." International Journal of Innovative Technology and Exploring Engineering 9.11 (2020): 66-73.
[5] Mendu, Mruthyunjaya, et al. "Development of real time data analytics based web applications using NoSQL databases." AIP Conference Proceedings. Vol. 2418. No. 1. AIP Publishing, 2022.
[6] Zanella, Andrea, et al. "Internet of things for smart cities." IEEE Internet of Things journal 1.1 (2014): 22-32.
[7] Porter, Michael E., and James E. Heppelmann. "How smart, connected products are transforming competition." Harvard business review 92.11 (2014): 64-88.
[8] Topol, Eric J. "High-performance medicine: the convergence of human and artificial intelligence." Nature medicine 25.1 (2019): 44-56.
[9] Armbrust, Michael, et al. "A view of cloud computing." Communications of the ACM 53.4 (2010): 50-58.
[10] Ian Gibson, Ian Gibson. "Additive manufacturing technologies 3D printing, rapid prototyping, and direct digital manufacturing." (2015).
[11] Yin, Kun, et al. "Advanced liquid crystal devices for augmented reality and virtual reality displays: principles and applications." Light: Science & Applications 11.1 (2022): 161.
[12] Lin, Ji, et al. "Mcunet: Tiny deep learning on iot devices." Advances in neural information processing systems 33 (2020): 11711-11722.
[13] Lee, Jay, Behrad Bagheri, and Hung-An Kao. "A cyber-physical systems architecture for industry 4.0-based manufacturing systems." Manufacturing letters 3 (2015): 18-23.
[14] Shehadeh, Maha, et al. "Digital transformation and competitive advantage in the service sector: a moderated-mediation model." Sustainability 15.3 (2023): 2077.
[15] Soori, Mohsen, Behrooz Arezoo, and Roza Dastres. "Internet of things for smart factories in industry 4.0, a review." Internet of Things and Cyber-Physical Systems 3 (2023): 192-204.
[16] Saberironaghi, Alireza, Jing Ren, and Moustafa El-Gindy. "Defect detection methods for industrial products using deep learning techniques: A review." Algorithms 16.2 (2023): 95.
[17] Abdul-Rashid, Salwa Hanim, et al. "Modelling sustainable manufacturing practices effects on sustainable performance: The contingent role of ownership." The International Journal of Advanced Manufacturing Technology 122.9 (2022): 3997-4012.
[18] Lee, Jay, Hung-An Kao, and Shanhu Yang. "Service innovation and smart analytics for industry 4.0 and big data environment." Procedia cirp 16 (2014): 3-8.
[19] Chen, Hsinchun, Roger HL Chiang, and Veda C. Storey. "Business intelligence and analytics: From big data to big impact." MIS quarterly (2012): 1165-1188.
[20] Bhattacheryay, Suranjan. "Multinational enterprises motivational factors in capitalizing emerging market opportunities and preparedness of India." Journal of Financial Economic Policy 12.4 (2020): 609-640.
[21] Hassoun, Abdo, et al. "The fourth industrial revolution in the food industry—Part I: Industry 4.0 technologies." Critical Reviews in Food Science and Nutrition 63.23 (2023): 6547-6563.
[22] Sham, Nabila Mohamad, and Azlinah Mohamed. "Climate change sentiment analysis using lexicon, machine learning and hybrid approaches." Sustainability 14.8 (2022): 4723.
[23] Raja Santhi, Abirami, and Padmakumar Muthuswamy. "Industry 5.0 or industry 4.0 S? Introduction to industry 4.0 and a peek into the prospective industry 5.0 technologies." International Journal on Interactive Design and Manufacturing (IJIDeM) 17.2 (2023): 947-979.
[24] Reddy, Pritika, Kaylash Chaudhary, and Shamina Hussein. "A digital literacy model to narrow the digital literacy skills gap." Heliyon 9.4 (2023).
[25] Haricha, Karim, et al. "Recent technological progress to empower smart manufacturing: Review and potential guidelines." IEEE Access 11 (2023): 77929-77951.
[26] Wylde, Vinden, et al. "Cybersecurity, data privacy and blockchain: A review." SN computer science 3.2 (2022): 127.
[27] Chi, Hao Ran, et al. "A survey of network automation for industrial internet-of-things toward industry 5.0." IEEE Transactions on Industrial Informatics 19.2 (2022): 2065-2077.

[28] Leng, Jiewu, et al. "Industry 5.0: Prospect and retrospect." Journal of Manufacturing Systems 65 (2022): 279-295. [13] Lee, Jay, Behrad Bagheri, and Hung-An Kao. "A cyber-physical systems architecture for industry 4.0-based manufacturing systems." Manufacturing letters 3 (2015): 18-23.
[29] Pinedo, Michael L., and Michael L. Pinedo. "Design and Implementation of Scheduling Systems: More Advanced Concepts." Scheduling: Theory, Algorithms, and Systems (2016): 485-508.
[30] Richey Jr, Robert Glenn, et al. "Artificial intelligence in logistics and supply chain management: A primer and roadmap for research." Journal of Business Logistics 44.4 (2023): 532-549.
[31] Chen, Zhongfei, and Guanxia Xie. "ESG disclosure and financial performance: Moderating role of ESG investors." International Review of Financial Analysis 83 (2022): 102291.
[32] Qian, Bin, et al. "A matrix-cube-based estimation of distribution algorithm for no-wait flow-shop scheduling with sequence-dependent setup times and release times." IEEE Transactions on Systems, Man, and Cybernetics: Systems 53.3 (2022): 1492-1503.
[33] Nahmias, Steven, and Ye Cheng. Production and operations analysis. Vol. 6. New York: McGraw-hill, 2009.
[34] Mamane, Asmae, et al. "Scheduling algorithms for 5G networks and beyond: Classification and survey." IEEe Access 10 (2022): 51643-51661.
[35] Fang, Xi, et al. "Smart grid—The new and improved power grid: A survey." IEEE communications surveys & tutorials 14.4 (2011): 944-980.
[36] Wu, Li-Jiao, et al. "Real environment-aware multisource data-associated cold chain logistics scheduling: A multiple population-based multiobjective ant colony system approach." IEEE Transactions on Intelligent Transportation Systems 23.12 (2022): 23613-23627.
[37] Lemon, Katherine N., and Peter C. Verhoef. "Understanding customer experience throughout the customer journey." Journal of marketing 80.6 (2016): 69-96.
[38] Gupta, Diwakar, and Brian Denton. "Appointment scheduling in health care: Challenges and opportunities." IIE transactions 40.9 (2008): 800-819.
[39] Gawali, Mahendra Bhatu, and Subhash K. Shinde. "Task scheduling and resource allocation in cloud computing using a heuristic approach." Journal of Cloud Computing 7 (2018): 1-16.
[40] A.S. Jain 1, S. Meeran, 1998, Deterministic job-shop scheduling: Past, present and future, Department of Applied Physics and Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, DD1 4HN, UK
[41] Majid Abdolrazzagh-Nezhad, Salwani Abdulla, 2017, Job Shop Scheduling: Classification, Constraints and Objective Functions, World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering
[42] Pinedo, M.L., 2008, Scheduling: Theory, Algorithms, and Systems, Stern School of Business, New York University, NY, USA
[43] Muth and G. Thompson. Industrial Scheduling. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1963
[44] Johnson, S.M., 1954. Optimal two- and three-stage production schedules with set-up times included.
[45] Jackson, J.R., 1956. An extension of Johnson's result on job lot scheduling. Naval Research Logistics Quarterly 3 (3), 201± 203.
[46] Conway, R.W., Maxwell, W.L., Miller, L.W., 1967. Theory of Scheduling. Addison±Wesley, Reading, MA.
[47] Roy, B., Sussmann, B., 1964. Les problemes d'ordonnancement avec contraintes disjonctives. Note D.S. no. 9 bis, SEMA, Paris, France.
[48] Zhang, Xiaohong, et al. "An exact branch-and-bound algorithm for seru scheduling problems with sequence-dependent setup time." Soft Computing 27.10 (2023): 6415-6436.
[49] Alhasnawi, Bilal Naji, et al. "A new communication platform for smart EMS using a mixed-integer-linear-programming." Energy Systems (2023): 1-18.
[50] Bliek1ú, Christian, Pierre Bonami, and Andrea Lodi. "Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report." Proceedings of the twenty-sixth RAMP symposium. 2014.
[51] Pedroso, Joo Pedro. "Optimization with gurobi and python." INESC Porto and Universidade do Porto,, Porto, Portugal 1 (2011).
[52] Kramer, Oliver, and Oliver Kramer. Genetic algorithms. Springer International Publishing, 2017.
[53] Nikolaev, Alexander G., and Sheldon H. Jacobson. "Simulated annealing." Handbook of metaheuristics (2010): 1-39.
[54] Baker, Kenneth R., and Dan Trietsch. Principles of sequencing and scheduling. John Wiley & Sons, 2018.
[55] Rajendran, Chandrasekharan, and Oliver Holthaus. "A comparative study of dispatching rules in dynamic flowshops and jobshops." *European journal of operational research* 116.1 (1999): 156-170.

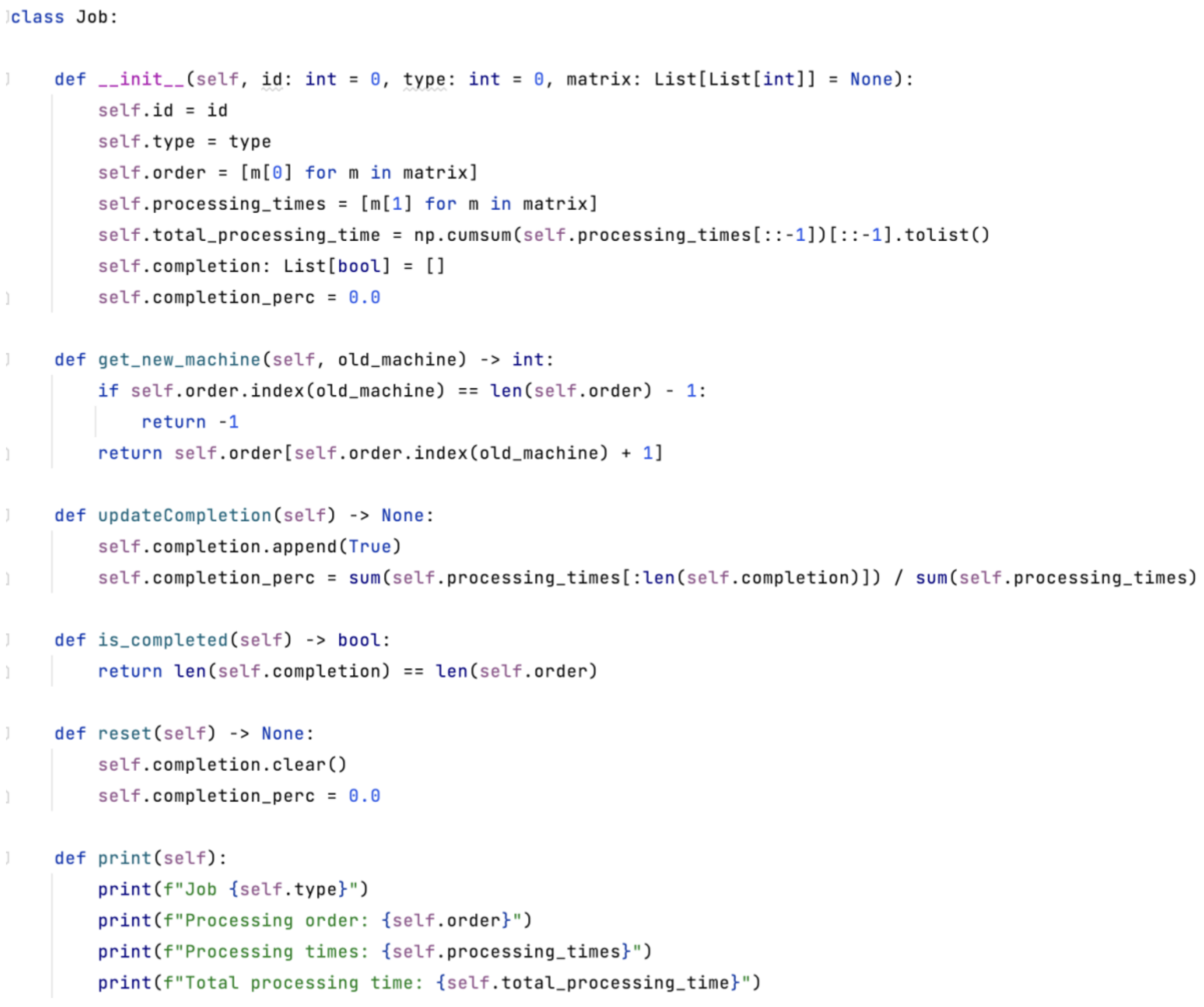
[56] Holland, John H. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press, 1992.
[57] McCarthy, John. "What is artificial intelligence." (2007): 2020.
[58] What is machine learning (ML)? IBM, URL: https://www.ibm.com/topics/machine-learning
[59] Richard S. Sutton and Andrew G. Barto, 2018, Reinforcement learning: An introduction, The MIT Press Cambridge, Massachusetts London, England
[60] Puterman, Martin L. Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons, 2014.
[61] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8 (1992): 279-292.
[62] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.
[63] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *nature* 521.7553 (2015): 436-444.
[64] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).
[65] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).
[66] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489
[67] Kabir, Golam, and M. Ahsan Akhtar Hasin. "Comparative analysis of artificial neural networks and neuro-fuzzy models for multicriteria demand forecasting." International Journal of Fuzzy System Applications (IJFSA) 3.1 (2013): 1-24.
[68]Introduction to Convolutional Neural Networks Architecture
https://www.projectpro.io/article/introduction-to-convolutional-neural-networks-algorithm-architecture/560
[69] Wu, Tianhao, et al. "Pairwise proximal policy optimization: Harnessing relative feedback for llm alignment." arXiv preprint arXiv:2310.00212 (2023).
[70] Irshayyid, Ali, and Jun Chen. "Comparative study of cooperative platoon merging control based on reinforcement learning." Sensors 23.2 (2023): 990.
[71] Liu, Z., Wang, Y., Liang, X., Ma, Y., Feng, Y., Cheng, G., & Liu, Z. (2022). A graph neural networks-based deep Q-learning approach for job shop scheduling problems in traffic management. Information Sciences, 607, 1211-1223.
[72] Tassel, P., Gebser, M., & Schekotihin, K. (2021). A reinforcement learning envi- ronment for job-shop scheduling. arXiv preprint arXiv:2104.03760.
[73] Moon J, Yang M, Jeong J. A Novel Approach to the Job Shop Scheduling Problem Based on the Deep Q-Network in a Cooperative Multi-Access Edge Computing Ecosystem. Sensors. 2021; 21(13):4553. https://doi.org/10.3390/s21134553
[74] Zhao,Y.,Wang,Y.,Tan,Y.,Zhang,J.,&Yu,H.(2021).Dynamicjobshopschedul- ing algorithm based on deep Q network. Ieee Access, 9, 122995-123011.
[75] Zhang, Zhiwei, et al. "A deep Q-network for job-shop scheduling in smart facto- ries." IEEE Transactions on Industrial Informatics 17.5 (2021): 3346-3354.
[76] Huang, S., & Ontañón, S. (2020). A closer look at invalid action masking in policy gradient algorithms. arXiv preprint arXiv:2006.14171.
[77] Huang, S., & Ontañón, S. (2020). A closer look at invalid action masking in policy gradient algorithms. arXiv preprint arXiv:2006.14171.
[78] Schulman,J.,Wolski,F.,Dhariwal,P.,Radford,A.,&Klimov,O.(2017).Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
[79] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. Jour- nal of Machine Learning Research, 22(268), 1-8.
[80] Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).
[81] Henderson, Peter, et al. "Deep reinforcement learning that matters." Proceedings of the AAAI conference on artificial intelligence. Vol. 32. No. 1. 2018
[82] Andrychowicz, Marcin, et al. "What matters in on-policy reinforcement learning? a large-scale empirical study." arXiv preprint arXiv:2006.05990 (2020).
[83] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
[84] Engstrom, Logan, et al. "Implementation matters in deep rl: A case study on ppo and trpo." International conference on learning representations. 2019.
[85] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[86] Bergstra, James, et al. "Algorithms for hyper-parameter optimization." Advances in neural information processing systems 24 (2011).
[87] Beasley, J. E., 1990, OR-Library, URL http://people. brunel.ac.uk/~mastjjb/jeb/info.html.
[88] B.M.OmbukietM.Ventresca,«LocalSearchGeneticAlgorithmsfortheJobShopSched- uling Problem », Applied Intelligence, p. 99-109, 2004.
[89] The Fourth Industrial Revolution is here — what makes it different? https://medium.com/@vmahesh/the-fourth-industrial-revolution-is-here-what-makes-it-different-8264720e88a2

Appendix

Dynamic Job Shop Scheduling RL Code

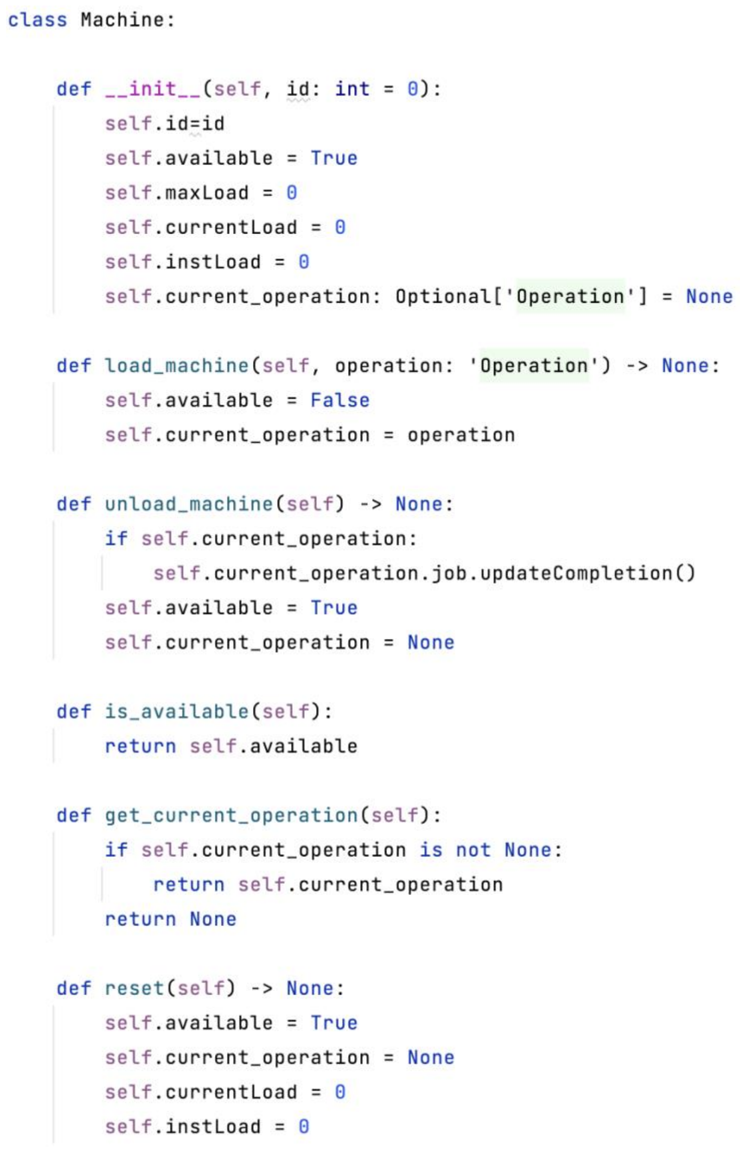
Job class



```python
class Job:

    def __init__(self, id: int = 0, type: int = 0, matrix: List[List[int]] = None):
        self.id = id
        self.type = type
        self.order = [m[0] for m in matrix]
        self.processing_times = [m[1] for m in matrix]
        self.total_processing_time = np.cumsum(self.processing_times[::-1])[::-1].tolist()
        self.completion: List[bool] = []
        self.completion_perc = 0.0

    def get_new_machine(self, old_machine) -> int:
        if self.order.index(old_machine) == len(self.order) - 1:
            return -1
        return self.order[self.order.index(old_machine) + 1]

    def updateCompletion(self) -> None:
        self.completion.append(True)
        self.completion_perc = sum(self.processing_times[:len(self.completion)]) / sum(self.processing_times)

    def is_completed(self) -> bool:
        return len(self.completion) == len(self.order)

    def reset(self) -> None:
        self.completion.clear()
        self.completion_perc = 0.0

    def print(self):
        print(f"Job {self.type}")
        print(f"Processing order: {self.order}")
        print(f"Processing times: {self.processing_times}")
        print(f"Total processing time: {self.total_processing_time}")
```

Figure 42: Job class

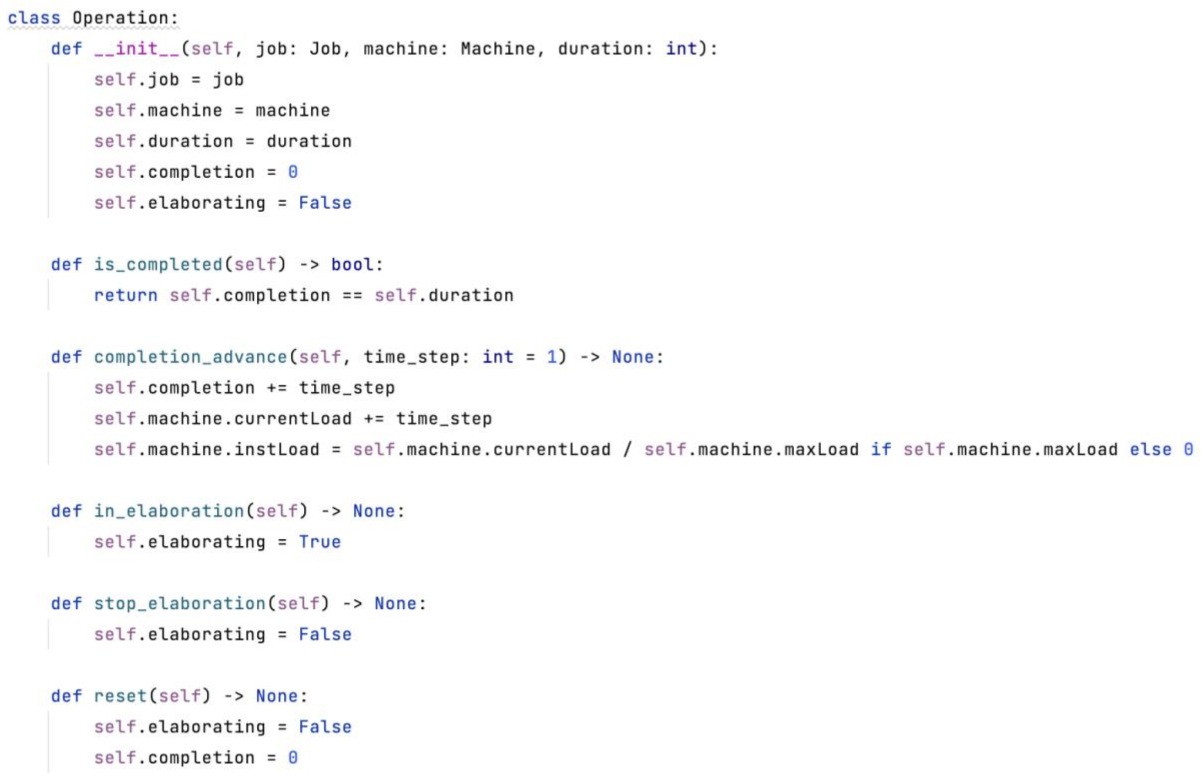
Machine class



```python
class Machine:

    def __init__(self, id: int = 0):
        self.id=id
        self.available = True
        self.maxLoad = 0
        self.currentLoad = 0
        self.instLoad = 0
        self.current_operation: Optional['Operation'] = None

    def load_machine(self, operation: 'Operation') -> None:
        self.available = False
        self.current_operation = operation

    def unload_machine(self) -> None:
        if self.current_operation:
            self.current_operation.job.updateCompletion()
        self.available = True
        self.current_operation = None

    def is_available(self):
        return self.available

    def get_current_operation(self):
        if self.current_operation is not None:
            return self.current_operation
        return None

    def reset(self) -> None:
        self.available = True
        self.current_operation = None
        self.currentLoad = 0
        self.instLoad = 0
```

Figure 43: Machine class

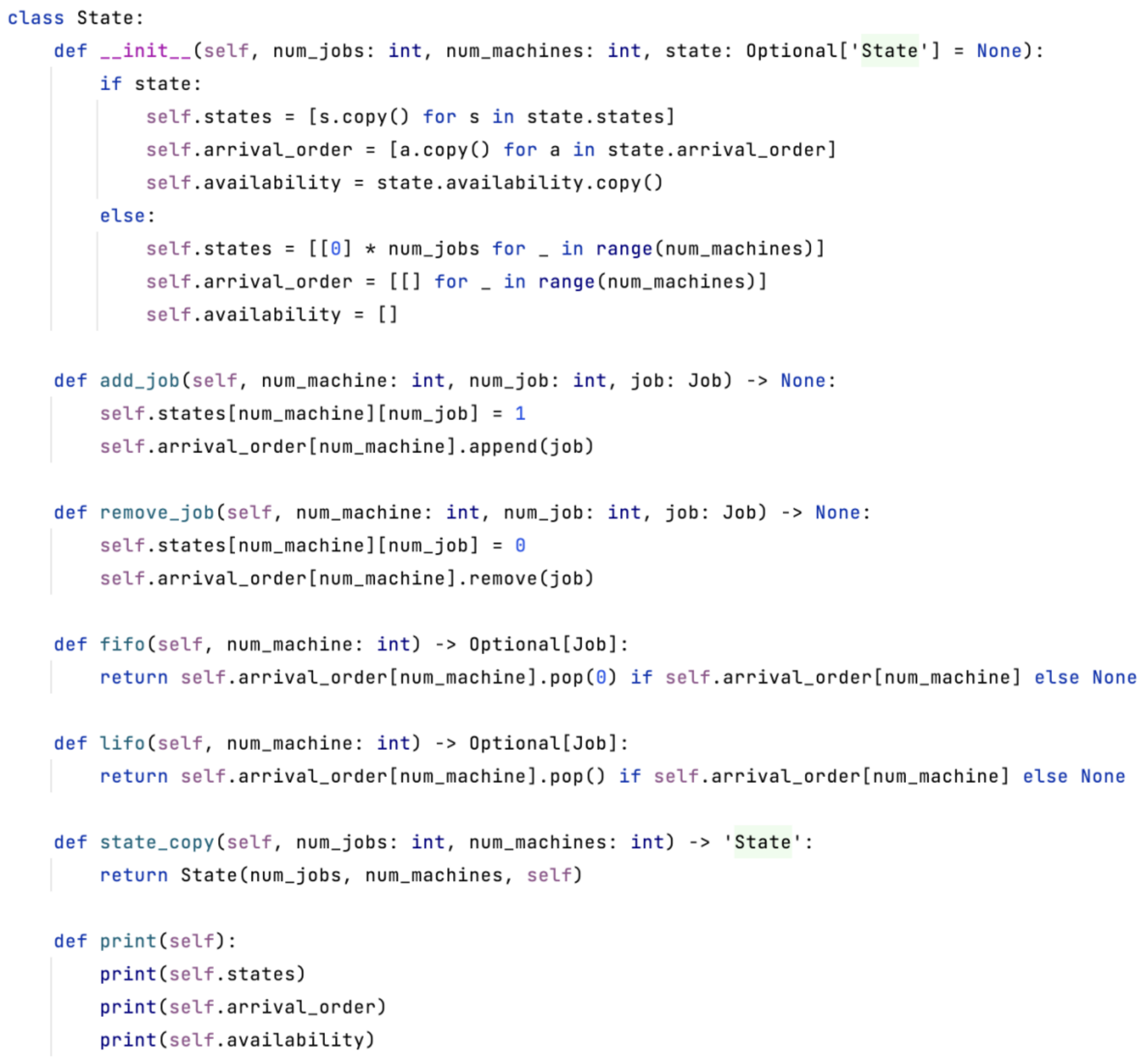
Operation class



```python
class Operation:
    def __init__(self, job: Job, machine: Machine, duration: int):
        self.job = job
        self.machine = machine
        self.duration = duration
        self.completion = 0
        self.elaborating = False

    def is_completed(self) -> bool:
        return self.completion == self.duration

    def completion_advance(self, time_step: int = 1) -> None:
        self.completion += time_step
        self.machine.currentLoad += time_step
        self.machine.instLoad = self.machine.currentLoad / self.machine.maxLoad if self.machine.maxLoad else 0

    def in_elaboration(self) -> None:
        self.elaborating = True

    def stop_elaboration(self) -> None:
        self.elaborating = False

    def reset(self) -> None:
        self.elaborating = False
        self.completion = 0
```

Figure 44: Operation class

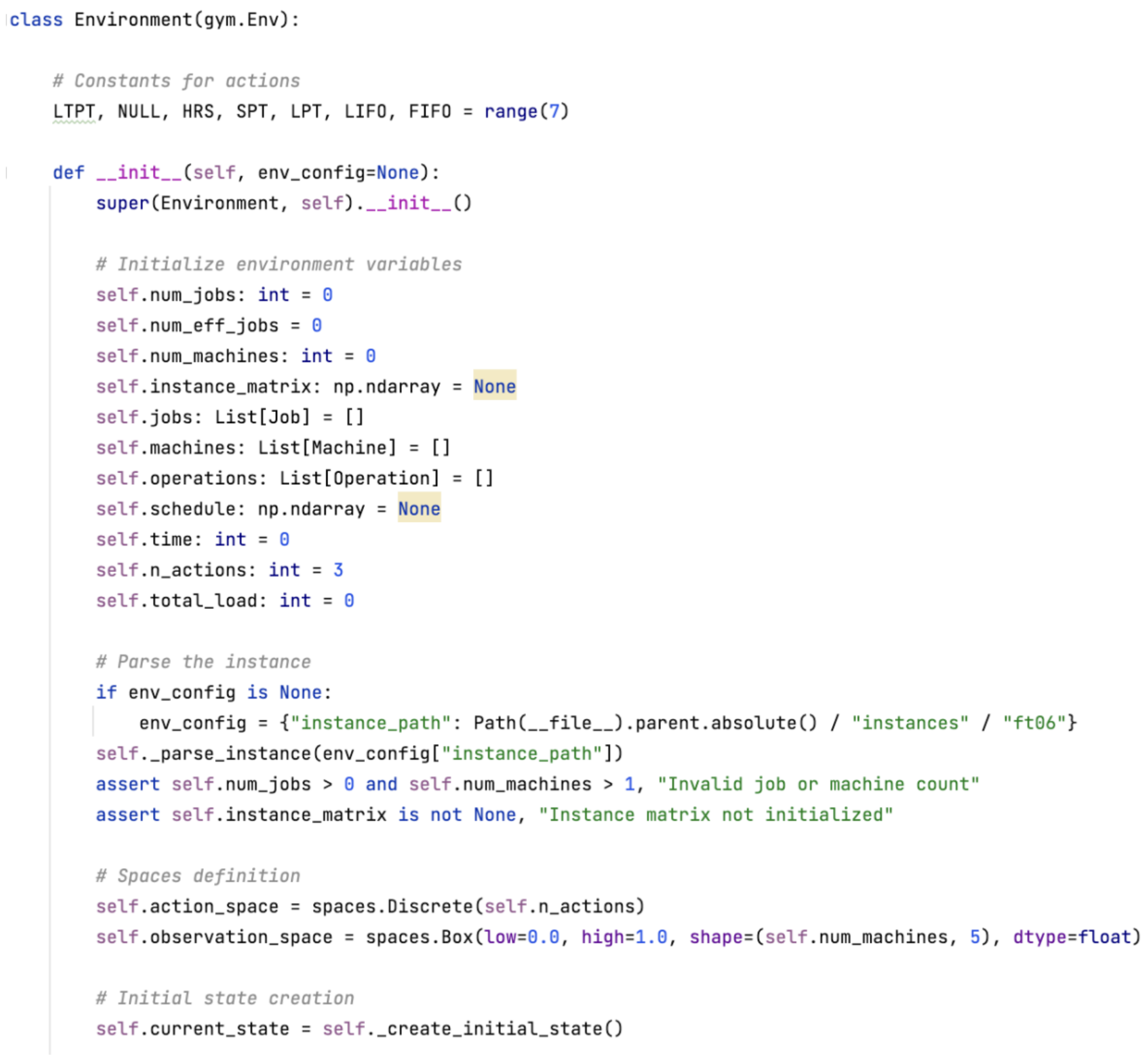
State class



```python
class State:
    def __init__(self, num_jobs: int, num_machines: int, state: Optional['State'] = None):
        if state:
            self.states = [s.copy() for s in state.states]
            self.arrival_order = [a.copy() for a in state.arrival_order]
            self.availability = state.availability.copy()
        else:
            self.states = [[0] * num_jobs for _ in range(num_machines)]
            self.arrival_order = [[] for _ in range(num_machines)]
            self.availability = []

    def add_job(self, num_machine: int, num_job: int, job: Job) -> None:
        self.states[num_machine][num_job] = 1
        self.arrival_order[num_machine].append(job)

    def remove_job(self, num_machine: int, num_job: int, job: Job) -> None:
        self.states[num_machine][num_job] = 0
        self.arrival_order[num_machine].remove(job)

    def fifo(self, num_machine: int) -> Optional[Job]:
        return self.arrival_order[num_machine].pop(0) if self.arrival_order[num_machine] else None

    def lifo(self, num_machine: int) -> Optional[Job]:
        return self.arrival_order[num_machine].pop() if self.arrival_order[num_machine] else None

    def state_copy(self, num_jobs: int, num_machines: int) -> 'State':
        return State(num_jobs, num_machines, self)

    def print(self):
        print(self.states)
        print(self.arrival_order)
        print(self.availability)
```

Figure 45: State class

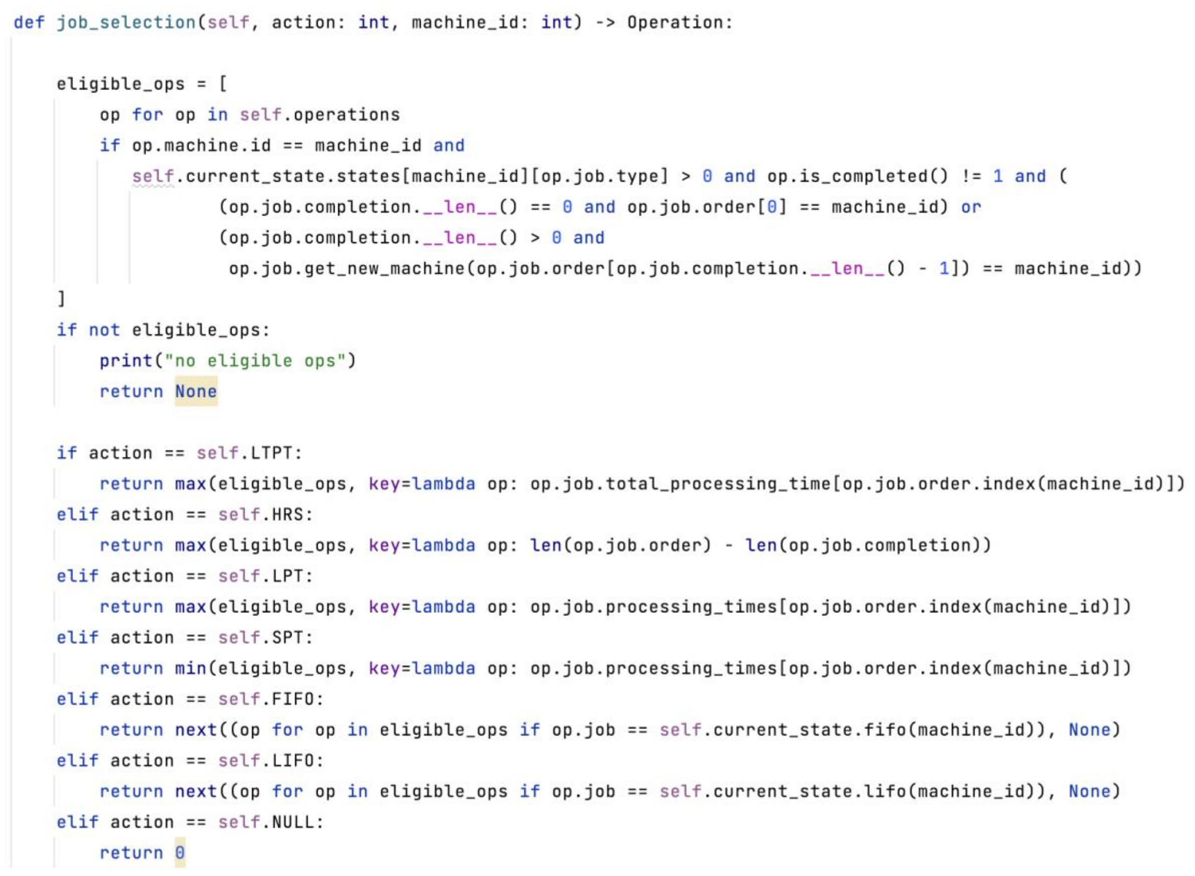
Environment class



```python
class Environment(gym.Env):

    # Constants for actions
    LTPT, NULL, HRS, SPT, LPT, LIFO, FIFO = range(7)

    def __init__(self, env_config=None):
        super(Environment, self).__init__()

        # Initialize environment variables
        self.num_jobs: int = 0
        self.num_eff_jobs = 0
        self.num_machines: int = 0
        self.instance_matrix: np.ndarray = None
        self.jobs: List[Job] = []
        self.machines: List[Machine] = []
        self.operations: List[Operation] = []
        self.schedule: np.ndarray = None
        self.time: int = 0
        self.n_actions: int = 3
        self.total_load: int = 0

        # Parse the instance
        if env_config is None:
            env_config = {"instance_path": Path(__file__).parent.absolute() / "instances" / "ft06"}
        self._parse_instance(env_config["instance_path"])
        assert self.num_jobs > 0 and self.num_machines > 1, "Invalid job or machine count"
        assert self.instance_matrix is not None, "Instance matrix not initialized"

        # Spaces definition
        self.action_space = spaces.Discrete(self.n_actions)
        self.observation_space = spaces.Box(low=0.0, high=1.0, shape=(self.num_machines, 5), dtype=float)

        # Initial state creation
        self.current_state = self._create_initial_state()
```

Figure 46: Environment class

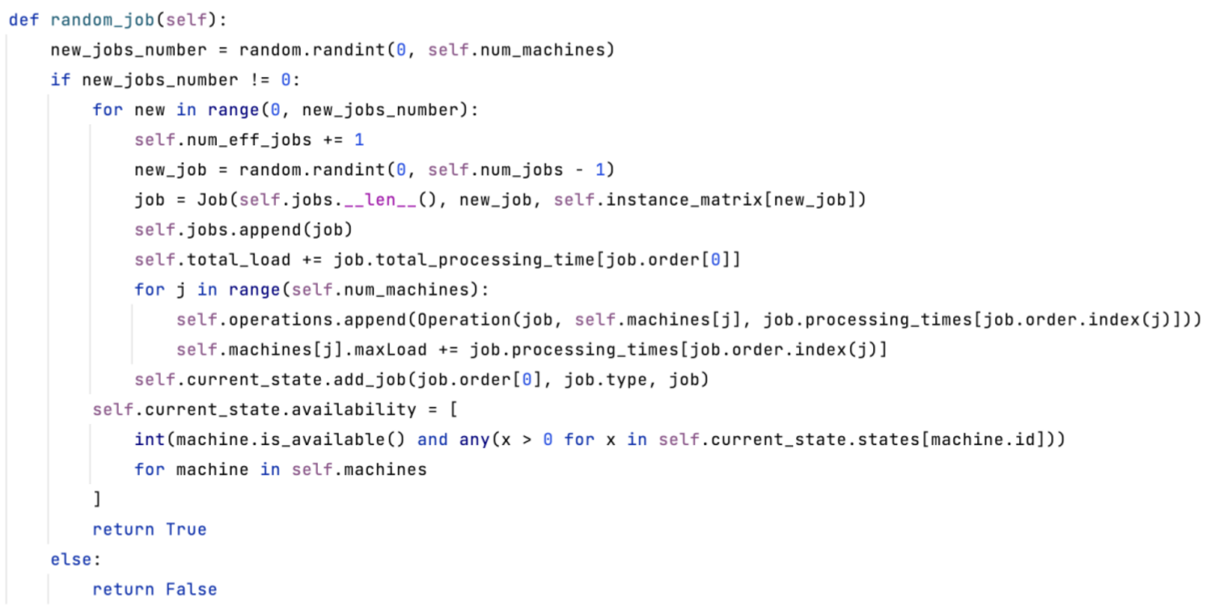
Job selection function



```python
def job_selection(self, action: int, machine_id: int) -> Operation:

    eligible_ops = [
        op for op in self.operations
        if op.machine.id == machine_id and
           self.current_state.states[machine_id][op.job.type] > 0 and op.is_completed() != 1 and (
                   (op.job.completion.__len__() == 0 and op.job.order[0] == machine_id) or
                   (op.job.completion.__len__() > 0 and
                    op.job.get_new_machine(op.job.order[op.job.completion.__len__() - 1]) == machine_id))
    ]
    if not eligible_ops:
        print("no eligible ops")
        return None

    if action == self.LTPT:
        return max(eligible_ops, key=lambda op: op.job.total_processing_time[op.job.order.index(machine_id)])
    elif action == self.HRS:
        return max(eligible_ops, key=lambda op: len(op.job.order) - len(op.job.completion))
    elif action == self.LPT:
        return max(eligible_ops, key=lambda op: op.job.processing_times[op.job.order.index(machine_id)])
    elif action == self.SPT:
        return min(eligible_ops, key=lambda op: op.job.processing_times[op.job.order.index(machine_id)])
    elif action == self.FIFO:
        return next((op for op in eligible_ops if op.job == self.current_state.fifo(machine_id)), None)
    elif action == self.LIFO:
        return next((op for op in eligible_ops if op.job == self.current_state.lifo(machine_id)), None)
    elif action == self.NULL:
        return 0
```

Figure 47: Job selection function

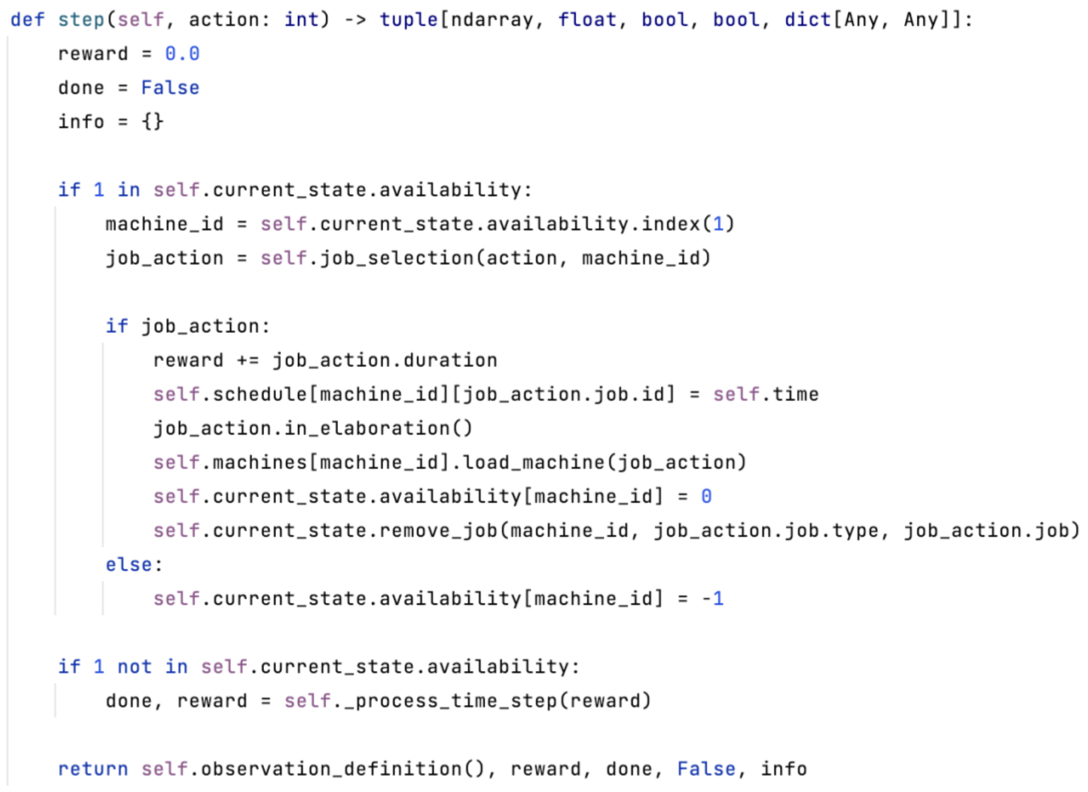
Random job arrival function



```python
def random_job(self):
    new_jobs_number = random.randint(0, self.num_machines)
    if new_jobs_number != 0:
        for new in range(0, new_jobs_number):
            self.num_eff_jobs += 1
            new_job = random.randint(0, self.num_jobs - 1)
            job = Job(self.jobs.__len__(), new_job, self.instance_matrix[new_job])
            self.jobs.append(job)
            self.total_load += job.total_processing_time[job.order[0]]
            for j in range(self.num_machines):
                self.operations.append(Operation(job, self.machines[j], job.processing_times[job.order.index(j)]))
                self.machines[j].maxLoad += job.processing_times[job.order.index(j)]
            self.current_state.add_job(job.order[0], job.type, job)
        self.current_state.availability = [
            int(machine.is_available() and any(x > 0 for x in self.current_state.states[machine.id]))
            for machine in self.machines
        ]
        return True
    else:
        return False
```

Figure 48: Random job arrival functio

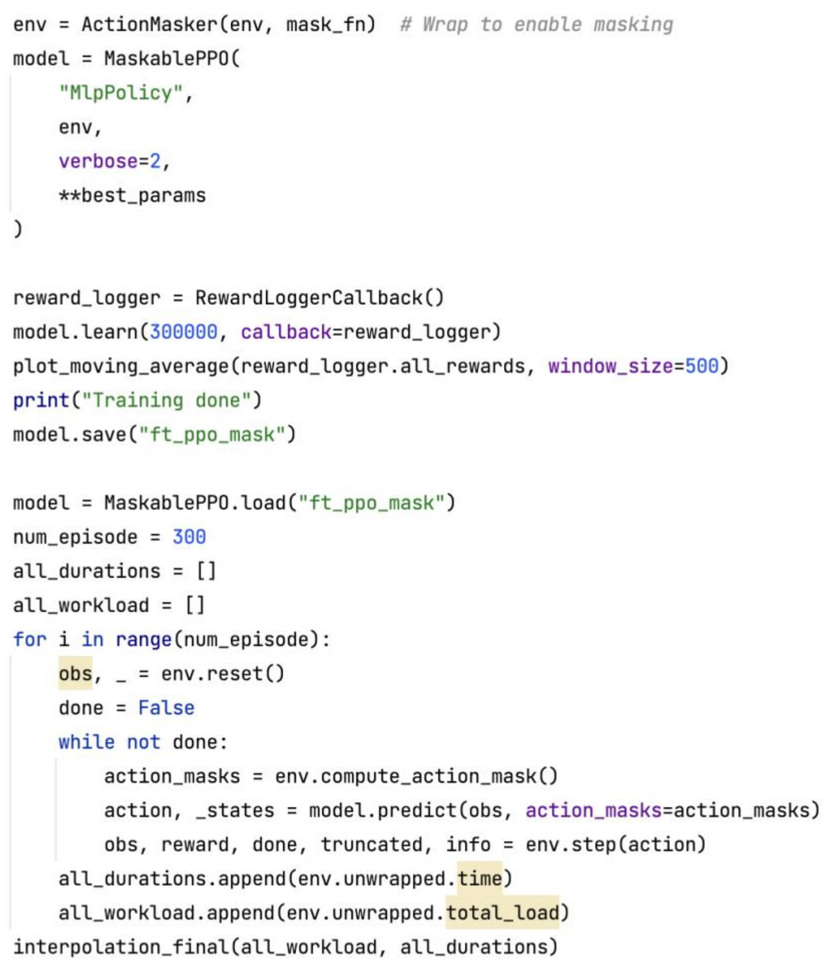
Step function



```python
def step(self, action: int) -> tuple[ndarray, float, bool, bool, dict[Any, Any]]:
    reward = 0.0
    done = False
    info = {}

    if 1 in self.current_state.availability:
        machine_id = self.current_state.availability.index(1)
        job_action = self.job_selection(action, machine_id)

        if job_action:
            reward += job_action.duration
            self.schedule[machine_id][job_action.job.id] = self.time
            job_action.in_elaboration()
            self.machines[machine_id].load_machine(job_action)
            self.current_state.availability[machine_id] = 0
            self.current_state.remove_job(machine_id, job_action.job.type, job_action.job)
        else:
            self.current_state.availability[machine_id] = -1

    if 1 not in self.current_state.availability:
        done, reward = self._process_time_step(reward)

    return self.observation_definition(), reward, done, False, info
```

Figure 49: Step function

Main

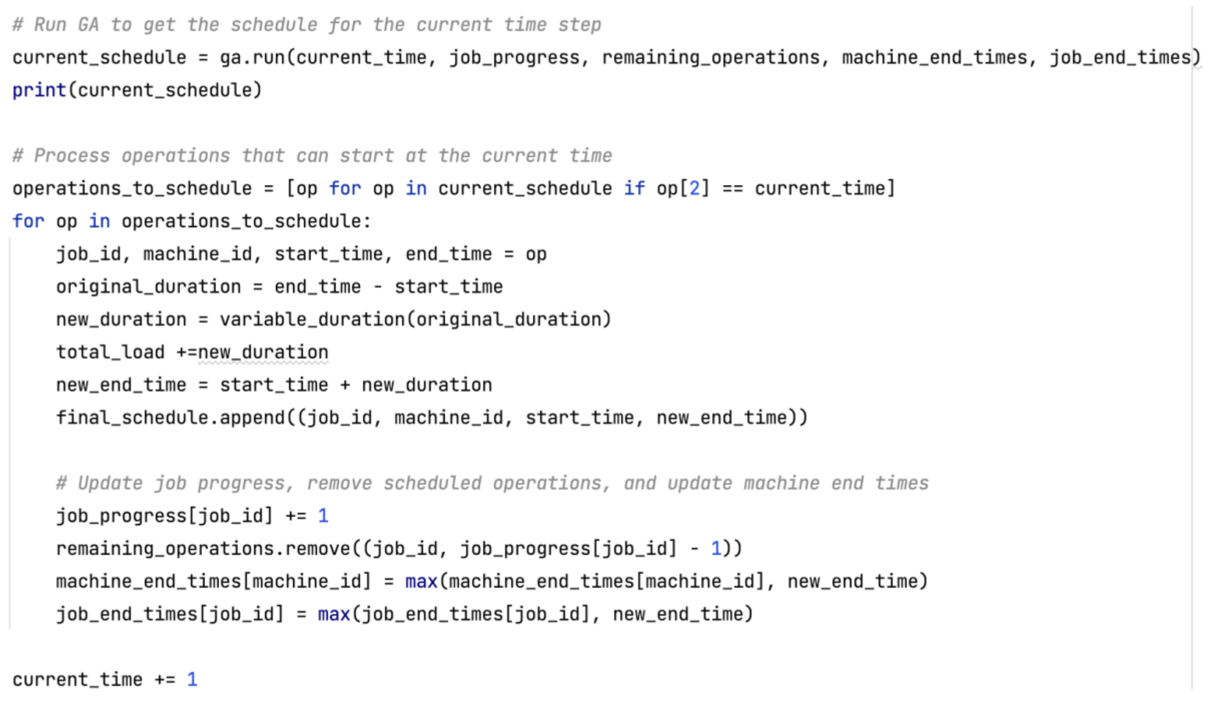


```
env = ActionMasker(env, mask_fn)  # Wrap to enable masking
model = MaskablePPO(
    "MlpPolicy",
    env,
    verbose=2,
    **best_params
)

reward_logger = RewardLoggerCallback()
model.learn(300000, callback=reward_logger)
plot_moving_average(reward_logger.all_rewards, window_size=500)
print("Training done")
model.save("ft_ppo_mask")

model = MaskablePPO.load("ft_ppo_mask")
num_episode = 300
all_durations = []
all_workload = []
for i in range(num_episode):
    obs, _ = env.reset()
    done = False
    while not done:
        action_masks = env.compute_action_mask()
        action, _states = model.predict(obs, action_masks=action_masks)
        obs, reward, done, truncated, info = env.step(action)
    all_durations.append(env.unwrapped.time)
    all_workload.append(env.unwrapped.total_load)
interpolation_final(all_workload, all_durations)
```

Figure 50: Main function

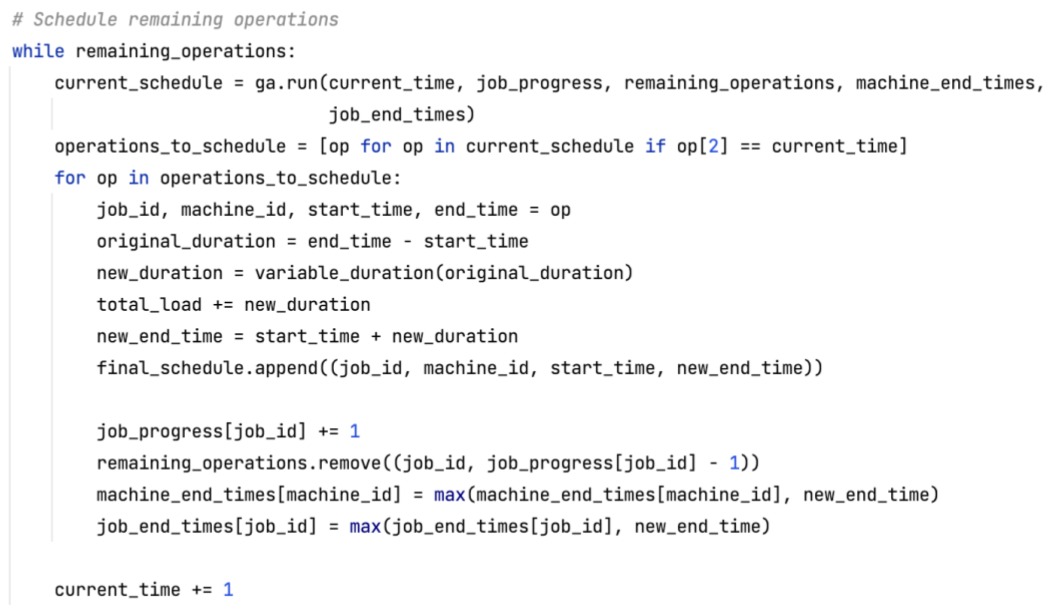
Dynamic Job Shop Scheduling Genetic Algorithm Code

Job handling function



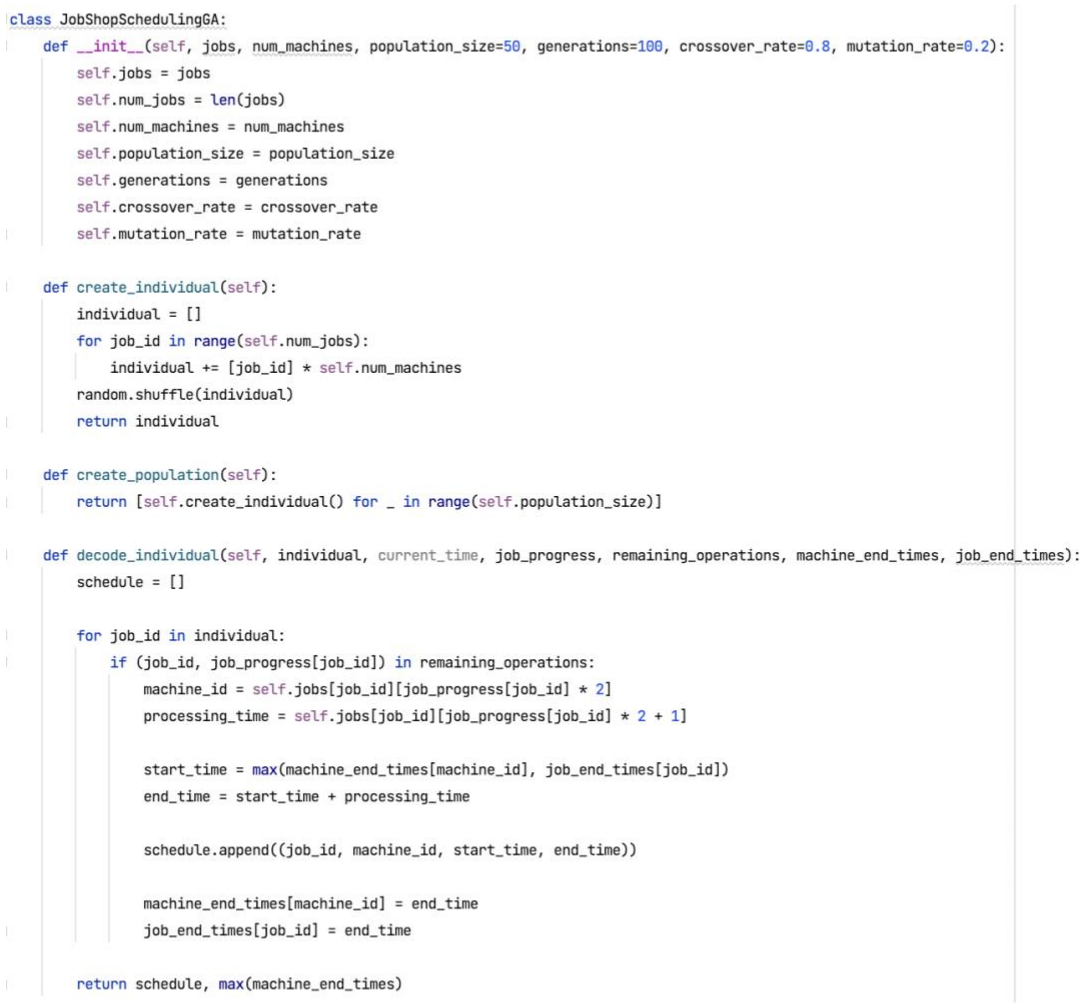
```python
# Run GA to get the schedule for the current time step
current_schedule = ga.run(current_time, job_progress, remaining_operations, machine_end_times, job_end_times)
print(current_schedule)

# Process operations that can start at the current time
operations_to_schedule = [op for op in current_schedule if op[2] == current_time]
for op in operations_to_schedule:
    job_id, machine_id, start_time, end_time = op
    original_duration = end_time - start_time
    new_duration = variable_duration(original_duration)
    total_load +=new_duration
    new_end_time = start_time + new_duration
    final_schedule.append((job_id, machine_id, start_time, new_end_time))

    # Update job progress, remove scheduled operations, and update machine end times
    job_progress[job_id] += 1
    remaining_operations.remove((job_id, job_progress[job_id] - 1))
    machine_end_times[machine_id] = max(machine_end_times[machine_id], new_end_time)
    job_end_times[job_id] = max(job_end_times[job_id], new_end_time)

current_time += 1
```

Figure 51: Job handling function (1)



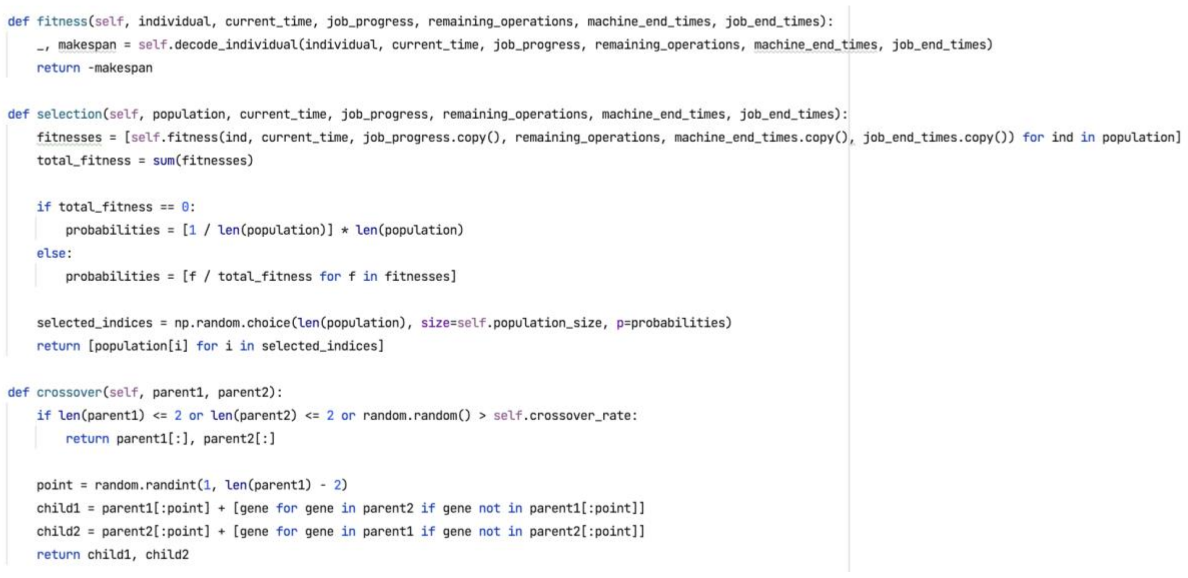
```python
# Schedule remaining operations
while remaining_operations:
    current_schedule = ga.run(current_time, job_progress, remaining_operations, machine_end_times,
                              job_end_times)
    operations_to_schedule = [op for op in current_schedule if op[2] == current_time]
    for op in operations_to_schedule:
        job_id, machine_id, start_time, end_time = op
        original_duration = end_time - start_time
        new_duration = variable_duration(original_duration)
        total_load += new_duration
        new_end_time = start_time + new_duration
        final_schedule.append((job_id, machine_id, start_time, new_end_time))

        job_progress[job_id] += 1
        remaining_operations.remove((job_id, job_progress[job_id] - 1))
        machine_end_times[machine_id] = max(machine_end_times[machine_id], new_end_time)
        job_end_times[job_id] = max(job_end_times[job_id], new_end_time)

    current_time += 1
```

Figure 52: Job handling function (2)



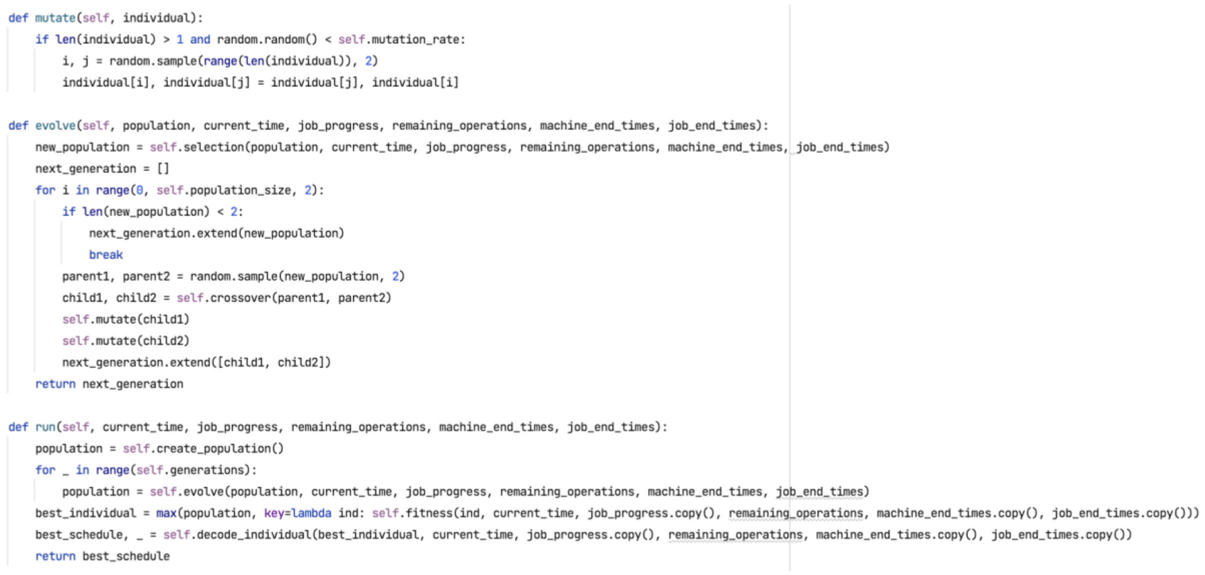
```python
class JobShopSchedulingGA:
    def __init__(self, jobs, num_machines, population_size=50, generations=100, crossover_rate=0.8, mutation_rate=0.2):
        self.jobs = jobs
        self.num_jobs = len(jobs)
        self.num_machines = num_machines
        self.population_size = population_size
        self.generations = generations
        self.crossover_rate = crossover_rate
        self.mutation_rate = mutation_rate

    def create_individual(self):
        individual = []
        for job_id in range(self.num_jobs):
            individual += [job_id] * self.num_machines
        random.shuffle(individual)
        return individual

    def create_population(self):
        return [self.create_individual() for _ in range(self.population_size)]

    def decode_individual(self, individual, current_time, job_progress, remaining_operations, machine_end_times, job_end_times):
        schedule = []

        for job_id in individual:
            if (job_id, job_progress[job_id]) in remaining_operations:
                machine_id = self.jobs[job_id][job_progress[job_id] * 2]
                processing_time = self.jobs[job_id][job_progress[job_id] * 2 + 1]

                start_time = max(machine_end_times[machine_id], job_end_times[job_id])
                end_time = start_time + processing_time

                schedule.append((job_id, machine_id, start_time, end_time))

                machine_end_times[machine_id] = end_time
                job_end_times[job_id] = end_time

        return schedule, max(machine_end_times)
```

Figure 53: Genetic algorithm class (1)



```python
def fitness(self, individual, current_time, job_progress, remaining_operations, machine_end_times, job_end_times):
    _, makespan = self.decode_individual(individual, current_time, job_progress, remaining_operations, machine_end_times, job_end_times)
    return -makespan

def selection(self, population, current_time, job_progress, remaining_operations, machine_end_times, job_end_times):
    fitnesses = [self.fitness(ind, current_time, job_progress.copy(), remaining_operations, machine_end_times.copy(), job_end_times.copy()) for ind in population]
    total_fitness = sum(fitnesses)

    if total_fitness == 0:
        probabilities = [1 / len(population)] * len(population)
    else:
        probabilities = [f / total_fitness for f in fitnesses]

    selected_indices = np.random.choice(len(population), size=self.population_size, p=probabilities)
    return [population[i] for i in selected_indices]

def crossover(self, parent1, parent2):
    if len(parent1) <= 2 or len(parent2) <= 2 or random.random() > self.crossover_rate:
        return parent1[:], parent2[:]

    point = random.randint(1, len(parent1) - 2)
    child1 = parent1[:point] + [gene for gene in parent2 if gene not in parent1[:point]]
    child2 = parent2[:point] + [gene for gene in parent1 if gene not in parent2[:point]]
    return child1, child2
```

Figure 54: Genetic algorithm class (2)



```
def mutate(self, individual):
    if len(individual) > 1 and random.random() < self.mutation_rate:
        i, j = random.sample(range(len(individual)), 2)
        individual[i], individual[j] = individual[j], individual[i]

def evolve(self, population, current_time, job_progress, remaining_operations, machine_end_times, job_end_times):
    new_population = self.selection(population, current_time, job_progress, remaining_operations, machine_end_times, job_end_times)
    next_generation = []
    for i in range(0, self.population_size, 2):
        if len(new_population) < 2:
            next_generation.extend(new_population)
            break
        parent1, parent2 = random.sample(new_population, 2)
        child1, child2 = self.crossover(parent1, parent2)
        self.mutate(child1)
        self.mutate(child2)
        next_generation.extend([child1, child2])
    return next_generation

def run(self, current_time, job_progress, remaining_operations, machine_end_times, job_end_times):
    population = self.create_population()
    for _ in range(self.generations):
        population = self.evolve(population, current_time, job_progress, remaining_operations, machine_end_times, job_end_times)
    best_individual = max(population, key=lambda ind: self.fitness(ind, current_time, job_progress.copy(), remaining_operations, machine_end_times.copy(), job_end_times.copy()))
    best_schedule, _ = self.decode_individual(best_individual, current_time, job_progress.copy(), remaining_operations, machine_end_times.copy(), job_end_times.copy())
    return best_schedule
```

Figure 55: Genetic algorithm class (3)

Deterministic environment solutions

la01 solution: 666

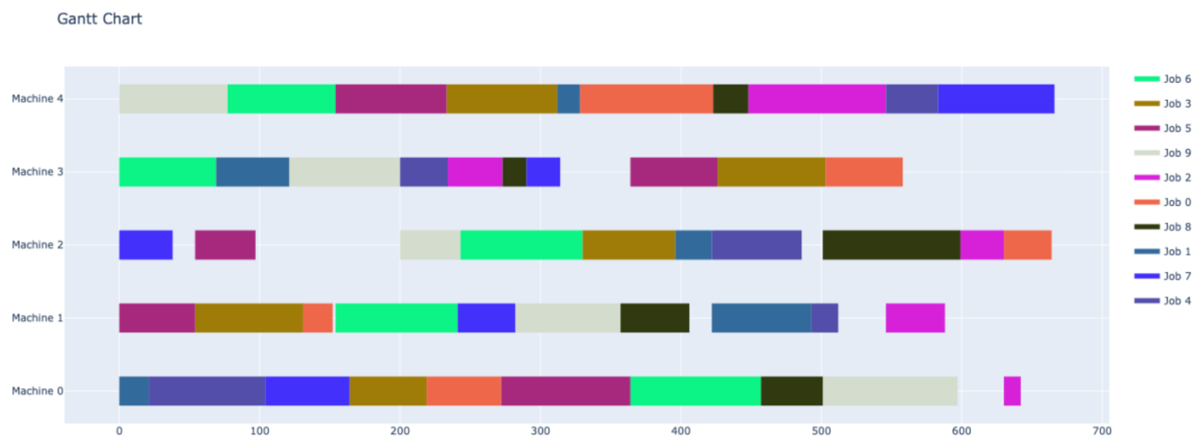


Figure 55: la01 solution

ft10 solution: 1032

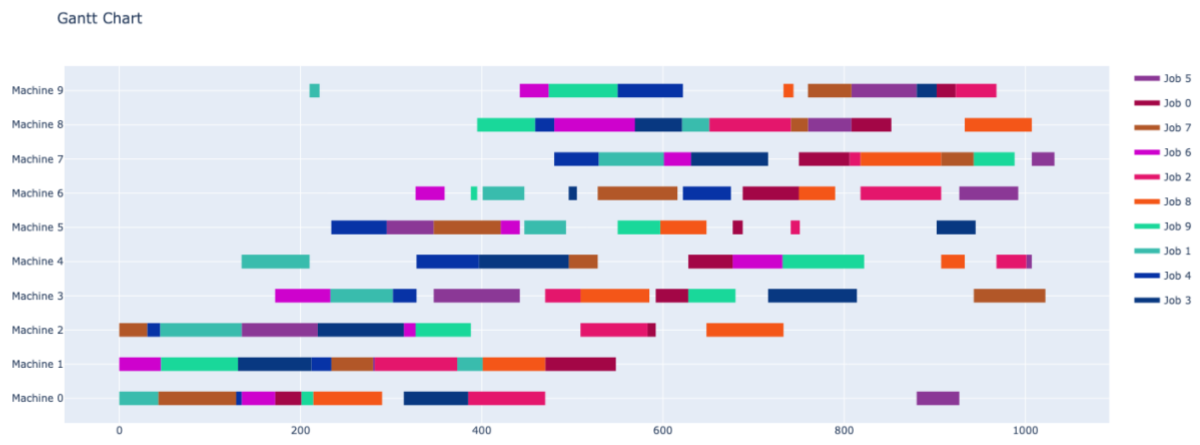


Figure 56: ft10 solution