Politecnico di Torino

Master of Science Degree in Engineering and Management
A.y. 2023/2024
Graduation Session: December 2024

A Deep Q-Network Approach to Job Shop Scheduling with Transport Resources

Academic Supervisor:
Prof: Arianna Alfieri
Prof: Erica Pastore
Prof: Claudio Castiglione

Candidate:
Julian Mateo Puerto Cortes

Academic Co-supervisor:
Prof: Gabriel Mauricio Zambrano Rey

Abstract

The Job Shop Scheduling Problem (JSSP) is a fundamental challenge in operations management, characterized by its NP-hard nature and the complexity of coordinating job sequences across multiple machines. This thesis addresses integrating transport resources, such as Automated Guided Vehicles (AGVs) and conveyor systems, into traditional JSSP, which adds a significant layer of complexity by introducing additional variables like transport times and availability constraints. A Deep Q-network (DQN) based methodology is developed to tackle these challenges, leveraging a neural network to approximate Q-values and guide decision-making in complex, stochastic environments. This approach is complemented by a Mixed Integer Linear Programming (MILP) formulation to establish foundational scheduling constraints. Through extensive simulation and benchmarking, this study demonstrates the effectiveness of DQN in optimizing scheduling in JSSP settings involving transport resources, contributing to both the theoretical understanding and practical implementation of machine learning-driven scheduling solutions in manufacturing contexts.

Contents

1. Introduction

The Job Shop Scheduling Problem (JSSP) is a core challenge in operations management. It is characterized by its NP-hard nature and the complexity of coordinating job sequences across multiple machines. JSSP involves allocating a set of jobs, each with specific processing requirements and sequences, to a limited set of machines. The goal is often to optimize performance metrics, such as makespan, machine utilization, and job tardiness. This problem is seen in manufacturing and production systems, where efficient scheduling directly impacts operational efficiency, productivity, and cost-effectiveness.

Introducing transport resources, such as Automated Guided Vehicles (AGVs) (Jungbluth et al., 2022; Amirteimoori et al., 2023; Fontes et al., 2023; Dehnavi-Arani et al., 2019) or mobile robots (Homayouni & Fontes, 2021; Yao et al., 2023; Li et al., 2020), adds complexity to traditional JSSP. These resources are essential for transferring jobs between machines, further complicating the scheduling process by introducing additional variables like transport times and availability constraints (Homayouni & Fontes, 2021). Integrating transport resources into scheduling models necessitates a dual optimization approach: minimizing machine idle times while ensuring timely material movement within the production line. This hybrid focus on machine processing and transport logistics reflects the complexity of modern job shop environments, where production efficiency is contingent upon effective material handling strategies (Fontes et al., 2022).

To address these challenges, recent research has turned to advanced machine learning techniques, particularly Reinforcement Learning (RL), which offers a dynamic and adaptive approach to scheduling. The Deep Q-Network (DQN) algorithm presents a promising solution among RL methods. DQN leverages a neural network to approximate Q-values, guiding decision-making in complex, stochastic environments. In this thesis, a DQN-based methodology is developed to optimize scheduling in JSSP settings involving transport

resources. This model allows the system to dynamically learn optimal policies for job sequencing and transport assignment, aiming to minimize makespan and streamline workflow.

This thesis integrates a structured Mixed Integer Linear Programming (MILP) formulation to establish foundational scheduling constraints, which serve as a baseline for the DQN environment—combining MILP and DQN bridges' exact and heuristic methods, balancing computational efficiency with solution quality. Through extensive simulation and benchmarking against traditional models, this study seeks to demonstrate the effectiveness of DQN in handling the intricacies of transport-integrated JSSP. This work thus contributes to both the theoretical understanding and practical implementation of machine learning-driven scheduling solutions in manufacturing contexts, laying the groundwork for scalable and adaptive job shop optimization strategies.

2. Literature review

2.1. Introduction

The Job Shop Scheduling Problem (JSSP) is a critical issue in operations management due to its significant impact on manufacturing efficiency and productivity (Muthiah et al., 2016). In a manufacturing environment, JSSP involves allocating jobs to machines where each job consists of a sequence of operations that must be processed on specific machines. The objective is to determine the optimal sequence in which these jobs should be executed to minimize various performance metrics such as makespan (Yao et al., 2023), total completion time (Li et al., 2020), and tardiness (Zambrano-Rey et al., 2023). The complexity of JSSP arises from the requirement that each job must be processed in a predetermined order and that each machine can handle only one operation at a time, leading to intricate interactions and dependencies among jobs. Classified as NP-hard, finding an optimal solution becomes computationally infeasible as the size of the problem increases, necessitating the development

of heuristic and metaheuristic approaches to find satisfactory solutions within a reasonable timeframe (Zhang et al., 2021).

The significance of JSSP in manufacturing and production systems cannot be overstated, as it directly influences operational efficiency, resource utilization, and overall productivity. Effective scheduling allows manufacturers to optimize production processes, reduce lead times, and enhance responsiveness to customer demands. In today's competitive manufacturing landscape, where customization and rapid delivery are paramount, efficient job scheduling is critical in aligning production schedules with market requirements. Moreover, it can lead to significant cost savings by minimizing idle time, reducing work-in-progress inventory, and maximizing machine utilization. As manufacturing systems evolve toward more flexible and dynamic operations, the importance of JSSP continues to grow, necessitating ongoing research and innovation in scheduling techniques (Zhang & Zheng, 2023; Momenikorbekandi & Abbod, 2023).

Transport resources are integral to effectively executing job shop scheduling, as they facilitate the movement of materials and components between different machines and workstations. These resources may include automated guided vehicles (AGVs) and manual handling equipment, all of which are essential for ensuring that jobs are processed promptly (Li et al., 2023). Integrating transport resources into JSSP introduces additional complexity, as scheduling must account not only for the processing times on machines but also for the travel times associated with moving materials. This necessitates the development of hybrid scheduling models that optimize both machine operations and transport logistics. By effectively managing transport resources, manufacturers can enhance workflow efficiency, reduce bottlenecks, apply defined scheduling rules, and improve overall production throughput (Han, 2023; Song et al., 2022).

Advancements in computational technology over recent years have significantly enhanced the ability to address complex scheduling problems. Observations such as Moore's Law, which notes the doubling of computational capacity approximately every two years, have facilitated developing and applying sophisticated algorithms and machine learning techniques to JSSP. This technological progress, particularly in the past five years, has accelerated research into new methodologies that leverage increased computational power to achieve more efficient and effective scheduling solutions.

This literature review aims to synthesize recent advancements in the field of job shop scheduling, focusing on the integration of transport resources and the application of various algorithmic techniques. By examining the evolution of scheduling methods over the past eight years—with an emphasis on the technological advancements in the most recent five years—we provide a comprehensive overview of the current state of research, highlighting the strengths and limitations of existing approaches. Furthermore, we explore the role of machine learning and reinforcement learning techniques in addressing the complexities of job shop scheduling, particularly their potential to enhance decision-making processes in dynamic and uncertain environments. Ultimately, this review seeks to identify gaps in the current body of knowledge and propose directions for future research that could lead to more effective and efficient scheduling solutions in manufacturing systems (Alabajee et al., 2020; Zhang et al., 2022; Awad & Abd-Elaziz, 2021; Zhao et al., 2021; Bozzi, 2023; Latthawanichphan et al., 2019).

2.2. Historical Context and Evolution of JSSP

The historical context of the Job Shop Scheduling Problem (JSSP) reveals a rich evolution of research and methodologies aimed at addressing its inherent complexities. Initially conceptualized in the 1950s, the JSSP has undergone significant transformations as manufacturing practices have evolved. Early studies primarily focused on deterministic

scheduling models, where job arrival times and processing durations were known in advance. However, as manufacturing environments became more dynamic and unpredictable, researchers began to explore flexible job-shop scheduling (FJSP) models that accommodate variations in job characteristics and machine capabilities. The introduction of metaheuristic approaches, such as genetic algorithms and simulated annealing, marked a pivotal shift in the field, enabling practitioners to tackle more extensive and more complex scheduling instances. Recent advancements have further integrated machine learning techniques, particularly reinforcement learning, into scheduling frameworks, allowing real-time adaptation to changing production conditions. This historical trajectory underscores the ongoing relevance of JSSP research in the context of modern manufacturing challenges as scholars continue to innovate and refine scheduling methodologies to meet the demands of increasingly complex production systems (Zeng & Wang, 2018; Zhang et al., 2021; Yu et al., 2020).

2.2.1. Early approaches to JSSP

Early approaches to the Job Shop Scheduling Problem (JSSP) laid the groundwork for understanding the complexities of scheduling jobs within manufacturing environments. Initially, researchers focused on deterministic models that assumed fixed processing times and machine availability, which allowed for the development of straightforward heuristic methods. One of the pioneering contributions was Johnson's algorithm, which provided optimal solutions for two-machine flow shop scheduling, setting a precedent for subsequent research in JSSP. However, as the complexity of manufacturing systems grew, these early methods proved inadequate for addressing the intricacies of real-world scheduling scenarios. The limitations of deterministic models prompted researchers to explore more sophisticated techniques, such as branch-and-bound algorithms and dynamic programming, which aimed to provide exact solutions for smaller instances of JSSP. Despite their effectiveness, these exact methods faced scalability challenges, leading to the exploration of heuristic and metaheuristic

approaches that could yield satisfactory solutions for larger and more complex job-shop scenarios. The evolution of these early approaches significantly impacted the development of more advanced scheduling methodologies, highlighting the need for flexibility and adaptability in modern manufacturing systems (Kirilov & Guliashki, 2017).

2.2.2. Development of flexible job shop scheduling (FJSP)

The development of Flexible Job Shop Scheduling (FJSP) marked a significant advancement in scheduling methodologies, addressing the limitations of traditional JSSP by allowing for greater flexibility in machine assignments and job processing sequences. Introduced in the early 1990s, FJSP acknowledges that modern manufacturing systems often require jobs to be processed on multiple machines, enabling manufacturers to optimize resource utilization and respond more effectively to changes in production requirements. This flexibility is particularly beneficial in environments where machine breakdowns or varying job priorities occur, as it allows for dynamic reallocation of resources. The FJSP problem is inherently more complex than the classic JSSP, as it introduces additional decision variables related to machine selection and job routing. Researchers have employed various optimization techniques, including genetic algorithms, simulated annealing, and tabu search, to tackle the complexities of FJSP. The growing interest in FJSP has led to a plethora of studies that explore hybrid approaches, combining traditional optimization methods with machine learning techniques to enhance scheduling performance in dynamic environments. This evolution has improved scheduling efficiency and contributed to the development of more resilient manufacturing systems capable of adapting to fluctuating market demands (Xia & Wu, 2005; Ba et al., 2016).

2.2.3. Introduction of transport resources in scheduling

Introducing transport resources in scheduling has become a critical consideration in optimizing job shop environments, as it directly impacts the efficiency of material handling

and job processing. Transport resources, such as automated guided vehicles (AGVs) and conveyor systems, facilitate the movement of materials between machines and workstations, influencing overall workflow and production throughput.

As manufacturing systems have evolved towards more integrated and automated operations, the need to incorporate transport logistics into scheduling models has gained prominence. Researchers have begun to develop hybrid scheduling frameworks that simultaneously optimize machine operations and transport logistics, recognizing that delays in material handling can lead to significant bottlenecks in production.

The integration of transport resources into scheduling algorithms not only enhances the efficiency of job processing but also improves responsiveness to changes in production schedules and resource availability. This shift towards a more holistic approach to scheduling reflects the complexities of modern manufacturing systems, where effective coordination between processing and transport resources is crucial for achieving operational excellence. The impact of incorporating transport resources into scheduling has been profound, leading to improved production efficiency and reduced lead times in various manufacturing contexts (Kumar et al., 2003; Ren et al., 2020).

2.3. Traditional Algorithms for JSSP

Traditional approaches to solving the Job Shop Scheduling Problem (JSSP) are divided into **exact** algorithms**,** heuristic methods**,** and metaheuristic approaches**,** each balancing solution quality and computational efficiency. Exact algorithms**,** such as branch-and-bound and integer programming, guarantee optimal solutions by exhaustively exploring the solution space. These methods are particularly useful in small-scale scenarios but face challenges in scalability due to the NP-hard nature of JSSP. For instance, branch-and-bound uses tree structures to evaluate and prune suboptimal solutions systematically, while integer programming formulates the problem as an optimization model to minimize performance

metrics under defined constraints (Muthiah et al., 2015; Lu et al., 2012). While effective for more minor problems, the exponential growth of the solution space in larger instances necessitates hybridization with heuristic methods to improve computational feasibility.

Heuristic methods prioritize speed and adaptability, making them particularly suitable for dynamic environments. Priority dispatching rules, such as Shortest Processing Time (SPT) and Earliest Due Date (EDD), are widely used to guide scheduling decisions based on specific criteria, enabling quick and practical solutions. Although these methods do not guarantee optimality, their simplicity and flexibility make them a strong foundation for more advanced algorithms (Amjad et al., 2018; Azzouz et al., 2017; Parveen & Ullah, 2011). Metaheuristic approaches, including simulated annealing and ant colony optimization, extend this flexibility by employing adaptive search strategies that balance exploration and exploitation of the solution space. Simulated annealing uses probabilistic acceptance criteria to escape local optima, while ant colony optimization leverages pheromone-based reinforcement learning to improve solutions iteratively. Both methods have proven effective in dynamic and complex scheduling contexts, with enhancements and hybridizations further boosting their performance (Xing et al., 2010; Fan & Su, 2022; Kumar et al., 2018; Sun et al., 2010).

These approaches collectively offer versatile solutions for JSSP, with exact algorithms excelling in precision, heuristics providing practicality, and metaheuristics achieving a balance between quality and scalability. Continued advancements in hybrid techniques and computational strategies are driving improvements in efficiency, adaptability, and the ability to address the diverse constraints of modern scheduling challenges.

2.4. Machine Learning Techniques in JSSP

Applying machine learning (ML) to the Job Shop Scheduling Problem (JSSP) has revolutionized scheduling processes, enabling adaptive and data-driven approaches to

optimize real-time decision-making. ML techniques such as supervised learning, unsupervised learning, and reinforcement learning (RL) allow systems to learn from historical data, adapt to dynamic conditions, and address the complexities of modern manufacturing environments. These methods significantly reduce computational complexity while enhancing the quality and responsiveness of scheduling solutions. RL has emerged as a potent tool, offering the ability to develop adaptive strategies through interactions with dynamic environments and enabling continuous improvement over time (Liu et al., 2020; Wang et al., 2019).

Supervised learning in JSSP involves training models on historically labeled data to predict optimal job assignments and sequences. Supervised learning effectively models the relationships between job characteristics and scheduling outcomes by using algorithms such as decision trees, neural networks, and support vector machines. These approaches can significantly improve efficiency and accuracy, though their success depends on the quality of the training data and their ability to generalize to unseen scenarios (Song et al., 2022; Chen et al., 2022). On the other hand, unsupervised learning identifies patterns and relationships within data without requiring labeled outputs. Techniques such as k-means clustering and Principal Component Analysis (PCA) are beneficial for grouping jobs with similar characteristics or reducing the dimensionality of complex scheduling data, which helps inform better scheduling decisions (Han & Yang, 2020; Tassel et al., 2022).

Reinforcement learning (RL) has proven exceptionally effective in addressing dynamic scheduling environments where job arrivals and machine availability fluctuate unpredictably. Unlike supervised or unsupervised learning, RL enables agents to interact with their environment, learning optimal strategies through cumulative rewards based on their actions. The development of deep reinforcement learning (DRL), which integrates neural networks with RL, has further expanded its capabilities, allowing for the modeling of complex state

spaces and the discovery of high-quality scheduling policies. DRL has been successfully applied to dynamic job shop scenarios, demonstrating its ability to adapt to changing conditions and optimize performance metrics like makespan and tardiness (Han & Yang, 2021; Zhao & Zhang, 2021). Additionally, attention mechanisms in DRL enhance scheduling models by focusing on critical features of the environment, improving interpretability and solution quality (Xu et al., 2022).

Recent advancements in multi-agent reinforcement learning (MARL) have addressed the challenges of coordinating multiple agents in complex scheduling scenarios. By enabling agents to share knowledge and learn cooperative strategies, MARL improves resource allocation and minimizes makespan in dynamic environments. For example, agents in MARL systems can effectively manage competing priorities in job-shop settings, optimizing their decisions based on collective experiences (Wu et al., 2021; Chen et al., 2022). Hybrid approaches combining RL with traditional optimization techniques, such as genetic algorithms or simulated annealing, further enhance scheduling outcomes by leveraging the complementary strengths of each method. These models allow for thoroughly exploring the solution space and refined decision-making processes, resulting in superior scheduling performance in complex job shop environments (Kim et al., 2022; Canese et al., 2021).

Despite these advancements, challenges remain in scaling ML techniques for large job shops, managing uncertainty in dynamic environments, and integrating transport resources into scheduling frameworks. As the number of jobs and machines increases, traditional algorithms often need help with the exponential growth of solution spaces, necessitating more scalable approaches (Ramasubbareddy et al., 2021; Zhou, 2024). Dynamic job shop settings introduce additional complexities, as job arrivals, processing times, and machine availability frequently fluctuate. Addressing these uncertainties requires algorithms capable of adapting to real-time changes while maintaining efficiency (Zhou et al., 2023; Lv, 2024). Furthermore,

integrating transport resources, such as automated guided vehicles (AGVs), into scheduling frameworks is critical for addressing bottlenecks and improving overall operational efficiency. Research has shown that optimizing production and transport processes together can significantly enhance scheduling performance (Li et al., 2022; Zhou et al., 2022).

2.5. Conclusion

The Job Shop Scheduling Problem (JSSP) remains a critical and intricate issue in operations management, significantly affecting manufacturing efficiency and productivity. Initially, research on JSSP concentrated on deterministic models, providing structured scheduling solutions suitable for controlled settings. However, as production requirements evolved, more dynamic scheduling models like Flexible Job Shop Scheduling (FJSP) emerged, allowing greater adaptability through the application of metaheuristic algorithms such as genetic algorithms, simulated annealing, and newer techniques like ant colony optimization (Zhang & Zheng, 2023; Kirilov & Guliashki, 2017).

Recent studies have emphasized integrating transport resources, such as automated guided vehicles (AGVs), into scheduling processes. While this integration adds complexity, it is essential for reducing lead times and avoiding bottlenecks, ultimately enhancing overall production efficiency (Kumar et al., 2003; Song et al., 2022).

Machine learning, especially reinforcement learning (RL), has emerged as a transformative approach to overcoming the limitations of traditional scheduling algorithms in JSSP. Techniques such as deep reinforcement learning (DRL) and hybrid RL models enable adaptive and responsive scheduling, particularly valuable in dynamic and uncertain conditions. These advancements demonstrate that RL-based methods can surpass traditional scheduling approaches by utilizing real-time data and learning from past scheduling patterns (Chang et al., 2022; Zhao & Zhang, 2021). By addressing scalability, adaptability, and resource integration challenges, machine learning techniques are driving significant

improvements in the efficiency and applicability of scheduling solutions in modern manufacturing systems.

Table 1.

Approaches taken to address the JSSP problem.

Technique	Description	Strength	Limitations	Applications / Focus Areas
Heuristic Methods	Use rule-based scheduling decisions, like priority dispatching (e.g., SPT, EDD).	Fast, low computational cost; suitable for quick, feasible solutions.	It may not reach the global optimum; it is sensitive to rule choice.	Real-time scheduling with quick decision needs.
Metaheuristic Approaches	Techniques that explore solution space adaptively, such as Genetic Algorithms and Simulated Annealing.	It can escape local optima and is effective in large-scale, complex scheduling.	Computationally intensive; requires tuning for optimal performance.	Large-scale JSSP with complex transport needs and resource constraints.
Exact Algorithms	Optimization methods like Integer Programming, Branch, and Bound guaranteed optimality.	Global optimality is useful in small-medium-sized problems with high precision needs.	Computationally infeasible for significant problems due to high complexity.	Suitable for small JSSPs or scenarios where optimality is critical.
Hybrid Models	Combine techniques, e.g., Genetic Algorithms with Simulated Annealing or RL with heuristics.	The balance between solution accuracy and computational efficiency is adaptable to different complexities.	Complex to design; performance depends on chosen combination and settings.	Flexible scheduling with complex constraints and dynamic resources.
Reinforcement Learning (RL)	Machine learning approach where an agent learns optimal scheduling policies via trial-and-error.	Learns adaptive policies over time; handles dynamic, real-time changes.	It requires extensive data/training and is challenging to model complex states and reward structures.	Dynamic JSSP where job or transport conditions change frequently.
Deep Reinforcement Learning (DRL)	RL enhanced with deep learning for handling high-dimensional state	Scalability and adaptability in complex, large JSSP instances;	High computational requirements; prone to	Large-scale JSSPs with complex transport and

	spaces (e.g., DQN, PPO).	effective in dynamic environments.	overfitting with limited data.	processing requirements.
Multi-Agent Systems (MAS)	Systems of agents representing resources (machines, vehicles) that cooperate in scheduling.	Flexibility in distributed decision-making; supports decentralized scheduling.	Coordination complexities may yield suboptimal global outcomes.	Distributed manufacturing systems with decentralized job and transport management.
Hybrid DRL & MAS	DRL combined with MAS for agent-based learning in complex scheduling environments.	It balances learning efficiency with decentralized decision-making and is suitable for distributed systems.	High computational cost: complex implementation and tuning required.	Distributed systems with multiple job and transport resources; adaptable scheduling.

3. Objective

This thesis aims to develop a comprehensive model for the Job Shop Scheduling Problem (JSSP) that optimizes both machine and transport resource allocation. The primary objectives are to minimize the makespan and enhance resource utilization, addressing the unique challenges posed by dynamic manufacturing environments. This research builds upon the frameworks established by Fontes et al. (2023), Ren et al. (2020), and Jungbluth et al. (2022), which approached similar problems using diverse algorithmic techniques and evaluated performance through makespan as the critical metric. Makespan, widely recognized as a benchmark for scheduling efficiency, serves not only as a measure of total completion time across all jobs but also as a foundation for deriving additional metrics such as lateness, which combines expected due dates with job completion times (Zambrano-Rey et al., 2023).

A critical component of this study is exploring the intrinsic relationship between resource utilization and makespan. Resource utilization, which measures the efficiency of deploying machines and transport resources, directly impacts the makespan by reducing idle times and mitigating bottlenecks. By ensuring optimal resource usage, this research hypothesizes that scheduling models can achieve significant reductions in makespan, thereby enabling faster

production cycles and improved operational performance. Through this integrated approach, the proposed model aims to enhance system efficiency and ensure its practical applicability in real-world, high-variability manufacturing contexts.

4. Methodology

After defining the aim of this thesis, the methodology will focus on using a hybrid approach to solve the JSSP with transport resources. This approach combines a mixed integer linear programming (MILP) model for core constraints with a Deep Q-network (DQN)- based Reinforcement Learning (RL) method to optimize dynamic scheduling. This integrated technique evaluates both static and dynamic problem aspects. The MILP model rigorously defines constraints and decision variables, providing a clear foundation for the RL environment. This model precisely structures job sequencing, machine availability, and Automated Guided Vehicle (AGV) transport operations as the defined transport resource based on literature and other benchmark instances as mentioned in the Literature Review.

The DQN-based RL solution is designed to minimize makespan and dynamically adjust to changes in job assignments and transport logistics. We benchmark our hybrid model's makespan, scalability, and resource efficiency through extensive experiments and simulations. The results indicate improvements in performance and adaptability compared to traditional methods, enhancing better results each time. Overall, the combination of MILP and DQN-based RL offers a robust framework for addressing the complexities of JSSP with transport resources, demonstrating the potential for enhanced efficiency and reduced operational costs.

4.1. Problem Setup and Formulation

The JSSP with transport resources involves assigning multiple jobs with specific machine sequences, transporting needs to machines (e.g., M1 to M4), and a Loading/Unloading (LU) area. Each machine handles only one job at a time, while AGVs move jobs between machines

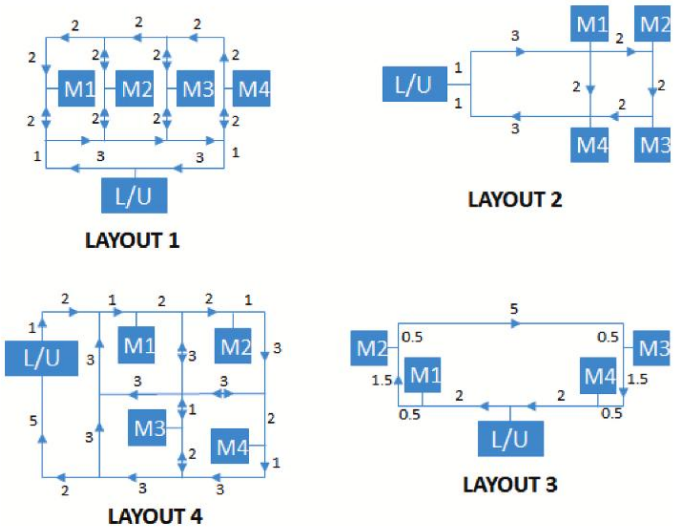
based on a predefined layout. The main objective is to minimize the makespan, which requires coordinating job scheduling, machine availability, and transport logistics under dynamic conditions. The MILP model was designed to formalize the core logic and constraints that ensure job sequencing, machine availability, and AGV transport operations. The model includes the following core components.

4.1.1. Decision Variables

Binary and continuous variables representing job assignments, machine and AGV allocations, and start/end times for each operation were defined to capture time costs associated with transport and machine operations (Azadeh et al., 2020).

Figure 1.

Transport times exposed by Bilge & Ulusoy (1995).



Note. Figure reproduced from *A multi-agent system simulation-based approach for collision avoidance in integrated job-shop scheduling problems with transportation tasks* (Abderrahim, 2023, p. 215). Reproduced under fair use.

https://doi.org/10.1016/j.jmsy.2023.03.011

4.1.2. Objective Function

The model's objective function minimizes makespan by calculating the time needed to complete all job sequences while reducing downtime and transport times. The makespan is

typically minimized alongside energy consumption to provide a multi-objective optimization model, where energy costs include the energy needed for AGV transport as well as for machine operations (Meng et al., 2023).

4.1.3. Constraints

- Job Sequencing: Constraints enforce that each job follows its predefined sequence, only starting subsequent operations after preceding operations are completed.
- Machine and AGV Constraints: Each machine can only process one job at a time, and each AGV can transport only one job between machines, echoing constraints often used in MILP models for transport-dependent scheduling (Azadeh et al., 2020; Meng et al., 2023).
- Transport Path Constraints: Transport between machines is scheduled based on layout-specific transport times, ensuring AGVs are managed efficiently and comply with machine location constraints.

These constraints were structured to guide the DQN environment, ensuring that the model respects essential scheduling principles and dynamically optimizes job completion within real-time constraints. While the MILP model was not executed due to computational requirements, it is the logical basis for implementing a constrained RL environment.

4.2. Deep Q-Network (DQN) Implementation

Following the MILP formulation, the RL-based approach is implemented using a DQN model, which is well-suited to real-time decision-making under the JSSP's complex state and action spaces. The DQN-based environment (JobShopEnv) simulates real-time scheduling with the following state components:

- **Machine Statuses**: Each machine's availability, tracking when it is free or occupied.
- **AGV Statuses and Locations**: Each AGV's availability, along with its current position, is updated as it moves between jobs.

- **Job-Machine Assignments**: Tracks the next required machine for each job based on its sequence.
- **Time Management**: The model tracks current time and machine availability times, facilitating state transitions based on progress updates.

Each element is dynamically updated, allowing the DQN model to learn and optimize scheduling through state-action pairs while minimizing the overall makespan.

The design and iterative refinement of the Deep Q-Network (DQN) model for the Job Shop Scheduling Problem (JSSP) reflects a systematic approach to addressing complex scheduling tasks with transport resources. Each model development phase contributed specific enhancements to the network architecture, data handling, and learning mechanisms, leading to a robust final model.

The DQN model began with a foundational setup: a two-layer neural network designed to approximate Q-values in an essential scheduling environment. This version aimed to establish a baseline architecture to evaluate machine-job assignments, utilizing a simple replay buffer and standard stochastic gradient descent (SGD) for optimization. Although effective for initial experimentation, this architecture was limited by its:

- **Simple Network Structure**: Consisting of only two hidden layers (128 and 64 neurons), the initial version was unable to generalize well across varying job sequences and transport requirements.
- **Single Layout Constraint**: Focused on a single layout configuration, this model had limited adaptability, which restricted its utility in diverse operational scenarios.

The initial version proved invaluable in identifying essential requirements for a more sophisticated network, explicitly highlighting the need for enhanced flexibility, improved state representation, and a scalable data-loading mechanism. Based on insights from this

model, the second version introduced architectural and functional enhancements to address its predecessor's limitations.

- **Expanded Network Architecture and RMSProp Optimization**: A third hidden layer was added to increase the model's interpretive capacity for complex job patterns, while the switch to RMSProp optimization provided better training stability, particularly in high-dimensional action spaces. Ending on a four-layer architecture able to handle more data robustness and identify optimal policies.
- **Integration of Azure Blob Storage**: The model now dynamically loads data for multiple layouts from Azure Blob Storage to handle more complex scheduling scenarios. This added adaptability allowed the model to support various configurations for transport and machine assignments, broadening its application scope and usability in real-world scenarios where data is not stored locally but on cloud services.

These modifications resulted in substantial gains in performance and stability. However, testing also revealed areas where the model's learning process could be further optimized, specifically by focusing on impactful transitions to improve decision-making. This insight led to the addition of a prioritized learning approach in the final version. The final DQN model stands as a culmination of iterative refinement, incorporating a sophisticated architecture and advanced training techniques that enable effective decision-making across diverse scheduling environments.

- **Four-Layer Neural Network with Prioritized Replay Buffer**: This final architecture includes four layers (with neurons of 256, 128, and 64), enhancing the model's ability to process high-dimensional state representations and complex scheduling patterns. Additionally, a prioritized replay buffer was implemented to

focus learning on significant transitions, which optimized the quality of the model's training and increased the precision of its scheduling decisions.

- **Double DQN Implementation for Stability**: By adopting Double DQN, the model addresses the issue of Q-value overestimation and achieves more reliable Q-value predictions. This enhancement proved particularly useful in dynamic environments where machine and AGV availability fluctuate, allowing the model to make more accurate decisions under varying conditions.

The development of this final model also incorporated an epsilon-decay strategy, which adjusted the exploration rate dynamically during training, balancing exploration and exploitation for optimal learning efficiency. Through these advancements, the final DQN model effectively manages job sequencing, machine assignments, and AGV allocations, demonstrating significant improvements in adaptability, stability, and overall performance.

The progression from a basic DQN framework to this sophisticated final model illustrates the importance of iterative experimentation and targeted adjustments. Each phase of development contributed to creating a DQN model equipped to meet the complex requirements of job shop scheduling with transport resources, reinforcing the value of adaptive, data-driven approaches in engineering scheduling solutions.

4.3. Performance Evaluation

The DQN's effectiveness is evaluated by comparing its makespan results against optimal values reported in previous studies, providing a basis for calculating the **optimality gap**. The optimality gap is measured as the difference between the DQN's makespan and the known optimal from other MILP-based methods. This comparison allows us to gauge the DQN's scheduling effectiveness relative to established optimization methods while respecting the scope and constraints initially laid out by the MILP model (Azadeh et al., 2020; Meng et al., 2023).

In sum, this methodology utilizes a hybrid approach that combines the structured constraint formulation of a MILP model with a flexible, real-time DQN-based RL solution. By grounding the DQN environment in MILP-defined constraints and benchmarking its results against established literature, this thesis provides a comprehensive framework for tackling the JSSP with transport resources in a way that balances optimization with real-world applicability.

5. MILP Model

In the following section, we analyze the Mixed Integer Linear Programming (MILP) model proposed by Fontes et al. (2023) to address the Job Shop Scheduling with Transport (JSPT) problem, known to be NP-hard. This model simultaneously tackles several interrelated combinatorial optimization challenges, specifically machine scheduling, vehicle assignment, and vehicle routing, effectively integrating them into a unified solution framework (Fontes et al., 2022). The importance of the model is that among the ones reviewed in the literature, the selected one includes the transportation times on the last transport task – from the previous machine to the Loading/Unloading (LU) area. This transport is vital because it allows for better calculation of the makespan. In the same way, these results are the most recent respect to the previous model published addressing the problem by the same authors.

The model's objective is twofold: (i) to obtain an optimal solution for more minor problem instances constrained by computational limits by minimizing expected job tardiness, and (ii) to ensure jobs return to the LU area as the final step in the transport schedule. As a result, each job operation requires a corresponding transport task to move the job to the designated machine, with the exception of the last task, which returns the job to the LU. To minimize exit time (makespan) for each job, a "dummy" operation with zero processing time is introduced for the final transport task. Upon returning to the LU, this operation signals the

job's completion. Thus, the completion time of this dummy operation represents the exit time for each job, marking when all necessary operations and transport tasks are concluded. This section details the specific notations, parameters, and decision variables used in this model, as structured by Fontes et al. (2023).

5.1. Sets and indices

J: Set of jobs, indexed by j and l.

M: Set of machines, indexed by m.

f, t: First and last dummy jobs, each composed of as many operations as the number of machines, respectively, the first and last operations of the machines that process them.

J_j: Set of n_j operations of the job $j \in J \cup \{f, t\}$, indexed by i and k.

$n_j + 1$: Dummy last operation of the job $j \in J$ processed at the LU (Loading unloading) with 0 processing time.

J'_j: Set of job operations $j \in J$ including the dummy operation, i.e., $J'_j = J_j \cup \{n_j + 1\}$.

5.2. Parameters

O_{ij}: Operation $i \in J'_j$ of job $j \in J$.

M_{ij}: Machine that processes operation $O_{ij}, i \in J'_j, j \in J$, where $M_{(n_j+1)j}$, is by definition, the LU.

T_{ij}: Transport task to deliver job j to machine $M_{ij}, i \in J'_j, j \in J$.

P_{ij}: Processing time of operation $O_{ij}, i \in J'_j, j \in J$ with $P_{(n_j+1)j} = 0$.

τ_{ij}^{kl}: Travel time from the machine M_{ij} to machine $M_{kl}, i \in J'_j, k \in J'_l, j, l \in J$.

A: Number of available vehicles.

LN: A sufficiently large positive integer.

5.3. Decision variables

w_{ij}^{kl}: Binary variable taking the value 1 if operation O_{kl} is processed immediately after the operation O_{ij} on the same machine and 0 otherwise, $k \in J_l, l \in J \cup \{t\}, i \in J_j, j \in J \cup \{f\}$.

x_{ij}^{kl}: Binary variable taking the value 1 if transport task T_{kl} is done immediately after transport task T_{ij} by the exact vehicle and 0 otherwise, $k \in J'_l, l \in J, T_{ij}, i \in J'_j, j \in J$.

u_{ij}: Binary variable taking the value 1 if transport task T_{ij} is the first task of a vehicle and 0 otherwise, $i \in J'_j, j \in J$

z_{ij}: Binary variable taking the value 1 if the transport task T_{ij} is the last task of a vehicle and 0 otherwise, $i \in J'_j, j \in J$.

5.4. Auxiliary variables

TD_j: Maximum tardiness expected for each job, $j \in J$.

c_{ij}: Completion time of operation $O_{ij}, i \in J'_j, j \in J$.

V_{ij}: Vehicle arrival time at the machine $M_{ij}, i \in J'_j, j \in J$.

5.5. Objective Function

Minimize ET (1)

5.6. Constraints

Subject to:

$$ET \geq c_{(n_l+1)l}, \ \forall l \in J \quad (2)$$

$$\sum_{i<k\in J_l} w_{il}^{kl} + \sum_{j\in J\setminus\{l\}} \sum_{i\in J_j} w_{ij}^{kl} + \sum_{i\in J_f} w_{if}^{kl} = 1, \ \forall l \in J \cup \{t\}, k \in J_l \quad (3)$$

$$\sum_{i>k\in J_l} w_{kl}^{il} + \sum_{j\in J\setminus\{l\}} \sum_{i\in J_j} w_{kl}^{ij} + \sum_{i\in J_t} w_{kl}^{it} = 1, \ \forall l \in J \cup \{f\}, k \in J_l \quad (4)$$

$$\sum_{j\in J} \sum_{i\in J'_j} z_{ij} = \sum_{j\in J} \sum_{i\in J'_j} u_{ij} \quad (5)$$

$$\sum_{j\in J} \sum_{i\in J'_j} u_{ij} \leq A \quad (6)$$

$$u_{kl} + \sum_{i<k\in J'_j} x_{il}^{kl} + \sum_{j\in J\setminus\{l\}} \sum_{i\in J'_j} x_{ij}^{kl} = 1, \ \forall l \in J, k \in J'_l \quad (7)$$

$$z_{kl} + \sum_{i>k\in J'_j} x_{kl}^{il} + \sum_{j\in J\setminus\{l\}} \sum_{i\in J'_j} x_{kl}^{ij} = 1, \ \forall l \in J, k \in J'_l \quad (8)$$

$$c_{kl} - v_{kl} - P_{kl} \geq 0, \ \forall l \in J, k \in J'_l \quad (9)$$

$$c_{kl} - c_{ij} - P_{kl} \geq LN\left(w_{ij}^{kl} - 1\right), \ \forall j, l \in J, i \in J_j, k \in J_l \ (\text{if } j = l : k > i) \quad (10)$$

$$v_{kl} - c_{(k-1)l} - \tau^{kl}_{(k-1)l} \geq 0,\ \forall l \in J, k \in J'_l \setminus \{1\} \quad (11)$$

$$v_{1l} - \tau^{1l}_{LU} \geq 0,\ \forall l \in J \quad (12)$$

$$v_{kl} - v_{ij} - \tau^{(k-1)l}_{ij} - \tau^{kl}_{(k-1)l} \geq LN\left(x^{kl}_{ij} - 1\right),\ \forall j, l \in J, i \in J'_l, k \in J'_l \setminus \{1\},\ (if\ j = l : k > i) \quad (13)$$

$$v_{1l} - v_{ij} - \tau^{LU\,l}_{ij} - \tau^{kl}_{LU\,l} \geq LN\left(x^{1l}_{ij} - 1\right),\ \forall j, l \in J : j \neq l, i \in J'_j \quad (14)$$

$$w^{kl}_{ij}, x^{kl}_{ij} \in \{0,1\},\ \forall j, l \in J \cup \{f, t\}, i \in J'_j, k \in J'_l \quad (15)$$

$$TD, c_{kl}, v_{kl} \geq 0,\ \forall l \in J, k \in J'_l \quad (16)$$

$$0 \leq u_{kl}, z_{kl} \leq 1,\ \forall l \in J, k \in J'_l \quad (17)$$

5.7. Development of the model

Based on the model presented by Fontes et al. (2023) and its way of elaborating on the multiple constraints, below will be described each one of them and its interpretation. The objective is to minimize the maximum exit time (ET), and its value is determined as the most extensive completion time of the last (dummy) operation over all of the jobs, as in inequality (2). Constraints (3) and (4) Impose that each operation be immediately followed and immediately preceded by precisely one other operation on the same machine. Constraint (5) ensures the same number of first and last tasks for the vehicles, whereas constraint (6) ensures it is at most the number of available vehicles simultaneously in the system. Constraints (7) and (8) impose each task to be immediately followed and immediately preceded by precisely one other task, respectively. Given that, a set of determined tasks is set through constraints (5) to (8), vehicles are handled implicitly. Hence, an additional index is not required.

For the completion time, two constraints must be satisfied. On the one hand, (9) indicates that an operation can only be finished after the job arrives at the machine processing it and its processing time has elapsed. On the other hand, (10) ensures that the completion of an operation requires the completion of the previous operation on the same

machine in addition to its own processing time in the current one. Likewise, a job can only arrive at a machine to have a specific operation processed after its previous operation has been completed ($c_{(k-1)l}$) . The job has already been transported from the machine of the previous operation to the current one ($\tau^{kl}_{(k-1)l}$) as stated in constraint (11), except for being the first job operation (k=1). In that case, constraint (12) ensures that the arrival time of the vehicle to the first operation must have completed the whole travel to the first machine where the job will be processed, as it will arrive as soon as the job has been transported from the L/U area to the first processing machine.

Furthermore, if a vehicle has transported some other job ($j \in J: j \neq l$)Immediately before the current one, then it needs to (i) deliver such a job to the corresponding machine (v_{ij}); (ii) pick up the current job from where the previous operation was processed ($\tau^{(k-1)l}_{ij}$) or the LU if it is the first operation (τ^{LU}_{ij}) and (iii) deliver it to the corresponding machine ($\tau^{ij}_{(k-1)l}$ or τ^{kl}_{LU}), enforced by constraints (13) and (14). Finally, from constraint (15) to (17) It defines the nature of the variables.

6. Deep Q-Network (DQN)

The Job Shop Scheduling Problem (JSSP) is a classic optimization problem in operations research, involving assigning jobs to resources (machines) over time, aiming to optimize performance measures such as makespan, total tardiness, or throughput. When transport resources, such as Automated Guided Vehicles (AGVs), are introduced to move jobs between machines, the complexity of the problem increases significantly. This added complexity necessitates advanced solution methods capable of handling the dynamic and stochastic nature of the environment.

Deep Reinforcement Learning (DRL) has emerged as a promising approach for solving complex scheduling problems due to its ability to learn optimal policies through interaction with the environment. Specifically, the Deep Q-Network (DQN) algorithm, introduced by

Mnih et al. (2015), combines Q-learning with deep neural networks to approximate the optimal action-value function.

The neural network-based job shop scheduling model evolved through three versions. Each iteration has progressively optimized both the environmental setup and training mechanisms to handle complex job scheduling scenarios involving multiple machines, automated guided vehicles (AGVs), and transport times, which are essential for minimizing makespan in dynamic scheduling environments.

6.1. Job Shop Scheduling Environment: Structural Setup and Data Organization

The **JobShopEnv** class, which simulates the scheduling environment, was designed to support 4 primary machines and a Loading-Unloading (LU) area, with multiple AGVs facilitating transport between areas. The environment's state space captures various elements:

- **Machine and AGV Statuses**: Tracking machine availability and AGV locations.
- **Job Sequences and Processing Times**: Each job follows a defined sequence of machines, with varying processing times and machine assignments.
- **Transport Times**: Each layout is defined by transport time matrices (df1, df2, df3, and df4) representing the time between locations for each of the four layouts.

Each version of the code improves the efficiency and accessibility of this data. By the final version, the **t_times () function** uses a dictionary to simplify the selection of transport times based on layout input, reducing redundancy.

6.2. Key Functions and Methods

1. **Environment Initialization and State Reset**

The environment is initialized with critical variables:

- machine_status: A binary array where 0 indicates a free machine, and 1 indicates a busy machine.

- agv_status and agv_locations: Track AGV availability and location to manage job transport.
- job_next_machine: Tracks each job's sequence to manage its progression through the job shop.

In each version, the reset () method reinitializes these variables at the beginning of an episode, preparing the environment for each new cycle. Given the hardware constraints, additional logging in later versions supports debugging and performance monitoring.

2. **State Representation and Sequence Completion Check**
   - **_get_state ()**: Concatenates data into a single state vector, including machine and AGV statuses, job-machine times, AGV locations, and locations.
   - **_check_done ()**: Verifies whether all jobs have completed their assigned machine sequences.
3. **Step Function and Transition Dynamics**

The **step(action)** method is central to the environment's dynamics. Each action is represented by a tuple (job, machine, AGV) that assigns a job to a machine with a specific AGV. The function validates machine and AGV availability, calculates transport and processing times, and updates the state.

The evolution of step () across versions highlights vital improvements:

- **Error Handling and Resource Checking**: In NNCodeRMS.py and NNCodeRMSV2.py, the method handles cases where resources are busy, advancing the environment's time to the following available resource and avoiding idle states.
- **Transport and Processing Time Calculation**:
  - **Transport Time (TT)**:

$$TT = T_{empty} + T_{loaded} \qquad (18)$$

- where:
  - T_{empty}: Transport time from the AGV's location to the job's location.
  - T_{loaded}: Transport time from the job's location to the target machine.
- **Processing Time (PT)**: Specific to each job-machine pair, calculated from data tables for each job's processing requirements on each machine.

These updates allow the model to track the dynamic scheduling scenario accurately, responding to job and machine availability in real-time.

6.3. Neural Network Architecture and Training Methodology

The neural network, **DQNScheduler**, is structured as a Deep Q-network (DQN) to approximate Q-values for each action-state pair. The DQN's architecture underwent enhancements to accommodate the increasing complexity of scheduling scenarios:

1. **Initial Model** Utilized a three-layer architecture with 128 and 64 neurons in hidden layers, optimized with the **Adam optimizer**.
2. **Expanded Model:** Increased hidden layer neurons to 256, 128, and 64, providing a more detailed mapping of the complex scheduling landscape. **RMSprop optimizer** replaced Adam to stabilize gradients, specifically under the sparse reward structure inherent in job shop scheduling.
3. **Final Model:** Incorporated a **Double DQN with a target network** to address the overestimation bias often present in Q-learning. By decoupling action selection from target generation, Double DQN improved the model's accuracy and convergence.

- **Target Network Update**: The target network's parameters are periodically synchronized with the leading DQN network, providing a stable target for Q-value updates and reducing instability.

6.4. Experience Replay and Prioritization

A critical component of the model's improvement is the **Replay Buffer**:

1. **Simple Replay Buffer (First and Second Versions)**: Stores recent experiences and samples them randomly for batch updates. The train_dqn_batch function selects a batch of experiences to compute the loss, defined as the **mean squared error (MSE)** between the predicted Q-values and target Q-values:

$$\text{loss} = \frac{1}{N}\sum_{i=1}^{N}\left(Q(s_i, a_i) - \left(r_i + \gamma \max_{a'} Q'(s_{i+!}, a')\right)\right)^2 \qquad (19)$$

where Q (s, a) is the predicted Q-value for state s and action a. γ is the discount factor, and Q' is the target network Q-value.

2. **Prioritized Replay Buffer (Final Version)**: In this version, introduced prioritized replay with adjustable prioritization (α) and importance sampling (β) to focus learning on high-error experiences, which likely represent more impactful transitions. This prioritization is managed by calculating the TD error for each experience:

$$\text{priority} = (|\text{TD error}| + \epsilon)^{\alpha} \qquad (19)$$

where α controls prioritization strength and ϵ avoids zero probability for any experience. Importance weights, w, adjust the loss for bias introduced by prioritized sampling:

$$w_i = \left(\frac{1}{N} * \frac{1}{P(i)}\right)^{\beta} \qquad (20)$$

where P(i) is the probability of sampling experience iii, and β\betaβ increases over time to reduce sampling bias.

6.5. Training the Model and Epsilon-Greedy Policy

Training dynamics evolved to optimize exploration-exploitation balance, with different epsilon decay strategies across versions:

1. **Early Exploration (First Version)**: The initial model maintained a constant decay rate to balance exploration and exploitation. The **train_dqn_batch** function updates the model with the sampled experiences.
2. **Adaptive Decay (Second and Final Versions)**: In these versions, decay rates adjust based on the training phase:
    - **Higher Decay Rates** in early episodes prioritize exploration.
    - **Reduced Decay Rates** in later episodes shift focus to exploitation, consolidating the learned policy.

The final version introduces the **Double DQN** structure, splitting Q-value maximization and action selection across two networks to avoid Q-value overestimation:

$$y = r + \gamma \, max_{a`} \, Q(s', a')y) \qquad (21)$$

where Q is the policy network, and Q′ is the target network.

6.5.1. Epsilon Decay greedy policy

The epsilon-greedy policy is a fundamental strategy in reinforcement learning that balances exploration (trying new actions) and exploitation (selecting the best-known actions based on learned Q-values). This balance is particularly crucial in job shop scheduling, where the model must efficiently search through a vast action space of job-machine-AGV assignments while quickly adapting to high-value scheduling patterns that minimize makespan.

1. **Epsilon (ε) Parameter and Decay Strategy**

The epsilon-greedy policy is controlled by an **epsilon parameter (ε)**, representing the probability that the model will select a random action rather than the action with the highest Q-value. Initially, epsilon is set high (e.g., ε = 1.0) to encourage exploration across diverse

actions. As training progresses, epsilon is gradually decayed to a lower threshold (e.g., $\epsilon_{min} = 0.1$), shifting the focus toward exploitation. The epsilon-greedy policy balances exploration and exploitation:

$$\pi(a|s) = \begin{cases} \text{random action}, & \text{with probability } \epsilon \\ \arg\max_a Q(s,a), & \text{with probaility } 1-\epsilon \end{cases} \quad (22)$$

In this way, the epsilon value decays over time to reduce exploration as the agent learns.

2. **Decaying Epsilon Across Episodes**

Each version of the model implemented a different epsilon decay strategy to optimize exploration-exploitation trade-offs:

- **Constant Decay Rate (First Version)**: In this version, epsilon decayed steadily, encouraging exploration in early episodes and moving to exploitation in later ones. However, this fixed rate limited flexibility in training adaptation.
- **Adaptive Decay Rate (Second and Final Versions)**: From the second version onwards, adapted the decay rate dynamically, depending on the training phase:
  - **Exploration Phase (First 90% of Episodes)**: As a primary basis, the focus was on exploring the solution space to focus on exploitation based on the possible solution at the end.
  - **Exploitation Phase (Last 10% of Episodes)**: Once the solution space was explored, it was essential to improve the possible results obtained.
- **Adaptive Decay Rate (Final Version)**: In this version, the decay rate was expanded dynamically across the phases:
  - **Explorative Phase (First 30% of Episodes)**: A slow decay rate (e.g., 0.999) maintained a high epsilon value to maximize exploration. This phase ensured a thorough search through potential job-machine-AGV configurations, helping the model learn the broader action space.

- **Balanced Phase (30-50% of Episodes)**: A moderate decay rate (e.g., 0.995) gradually reduced exploration, allowing the model to begin exploiting known high-reward actions while still discovering alternative strategies.
- **Exploitation Phase (Final 50% of Episodes)**: A faster decay rate (e.g., 0.990) minimized epsilon, steering the model toward exploitation. This phase consolidated learning, prioritizing the refinement of high-value strategies identified earlier.

3. **Action Selection and Behavior Adjustment**

   At each decision step, the model generates a random number between 0 and 1:

   - If the number is less than epsilon, the model performs **exploration** by selecting a random action. This prevents the model from getting trapped in suboptimal, repetitive schedules and helps it discover potential high-reward actions.
   - If the number is more significant than epsilon, the model performs **exploitation** by selecting the action with the highest Q-value for the current state. This Q-value is derived from the neural network's predictions, reflecting the cumulative expected reward for each action.

In the final version, the epsilon-greedy policy further benefited from **prioritized experience replay**. The model reinforced learning in challenging or impactful states by re-sampling experiences with high TD (temporal difference) errors, accelerating convergence to optimal job scheduling strategies even as epsilon decreased.

4. **Impact on Scheduling Performance**

The epsilon-greedy policy, especially with adaptive decay, played a critical role in balancing **initial exploration** with **later exploitation**. This approach improved the model's

ability to identify effective job-machine-AGV combinations, minimizing makespan across diverse layouts and job sets. By managing exploration strategically, the policy ensured that the final model could generalize across various scheduling environments and specialize in efficient, makespan-minimizing schedules as training progressed.

In summary, the epsilon-greedy policy's adaptive decay facilitated a progressive exploration-exploitation transition, critical to uncovering and solidifying high-quality scheduling policies. This strategy contributed to the model's robust learning curve, driving its capacity to minimize makespan effectively within the job shop environment.

6.6. Performance and Results

The progressive refinement across code versions produced a neural network model capable of:

- **Efficient Makespan Minimization**: Through dynamic action selection based on real-time state representation.
- **Scalable Action-Space Handling**: By incorporating prioritized experience replay and the Double DQN architecture, the model effectively learns optimal actions even in large, sparse action spaces.

The final version demonstrates robustness in handling high-dimensional scheduling tasks with real-time decision-making. The model achieved minimized makespan values, with improved stability and convergence facilitated by the prioritized replay buffer and target network architecture. Performance metrics were logged using TensorBoard, providing detailed insights into reward trajectories across training episodes.

Finally, this neural network-based scheduling model integrates complex operations research concepts within a deep learning framework, showcasing progressive improvements that advance both model robustness and computational efficiency. Each enhancement in neural network architecture and scheduling environment reinforces the model's capacity to

handle dynamic and realistic job shop scenarios with high precision and minimal computational overhead.

7. Results

Based on the model and results from various authors, this section will discuss the findings, compute the optimal gap between this model and others addressing the same issue, and detail the constraints encountered during development and testing. The reinforcement learning model was created in VS Code and deployed on a MacBook Pro with an M1 chip and 8GB of RAM. The optimal gap will be evaluated against the results in Table 3, using the following formula for percentage calculation:

$$Optimal\ gap = \frac{Value\ obtained\ on\ DQN\ Model}{Value\ obtained\ on\ MILP\ Model} - 1 \qquad (23)$$

This measure will be applied as a math formula to contrast the results. For this purpose, this will be evaluated on the benchmark instances created for the problem explained by Bilge & Ulusoy (1995); in the Literature Review section, Table 1 enumerates some

solutions deployed across the multiple papers enunciated to compare with.

The table below presents three versions of the model. The first uses the Adam optimizer and shows the best feasible time achieved. The second is a more advanced model with the RMSprop optimizer, applying the same methodology to preserve the results. The final version also uses the RMSprop optimizer but with bolder hyperparameters and a modified environment to ensure the MILP Model's presentation. Later, these results will be compared to those of the MILP Model and the times calculated from the following simplified Lower Bound calculated out of the following formulas:

$$C_j = \sum_{m \in M_j} p_{jm} + \sum_{m \in M_j, m' \to next\ (m)} t_{j,m \to m'} \qquad (24)$$

Where C_j is the completion time of a job, p_{jm} is the processing time of a job in the machines assigned to it, and $t_{j,m \to m'}$ is the transport time associated to the job from a

machine to the following one. The calculation is made for all the jobs in order to be compared against the following one:

$$T_{ls} = \max_{j \in J} C_j \qquad (25)$$

Where T_{ls} is the optimal time to be reeached when all resources are available or mostly commonly known as the Lower Bound (LB), indicating the minimum possible feasible time. In this scenario, this time includes the transport times from Loading/Unloading (LU) Area to the machines, and the last transport task to return the job to the LU Area.

Table 2.

Results for the BU instances to minimize the exit time (ET) for 2 AGVs.

Instance	Set	Layout	MILP	LB	PSOSA	ALS	MA
EX11	1	1	114	82	114	114	114
EX12	1	2	90	76	90	90	90
EX13	1	3	98	78	98	98	98
EX14	1	4	140	97	140	140	140
EX21	2	1	116	92	116	116	116
EX22	2	2	82	80	82	82	82
EX23	2	3	89	84	89	89	89
EX24	2	4	134	98	134	134	134
EX31	3	1	121	90	121	121	121
EX32	3	2	89	79	89	89	89
EX33	3	3	96	79	96	96	96
EX34	3	4	148	102	148	148	148
EX41	4	1	136	95	136	136	138
EX42	4	2	100	73	100	100	100
EX43	4	3	102	70	102	102	102
EX44	4	4	163	114	163	163	163
EX51	5	1	110	77	110	110	110
EX52	5	2	81	67	81	81	81
EX53	5	3	89	67	89	89	89
EX54	5	4	134	97	134	134	134
EX61	6	1	129	102	129	130	129
EX62	6	2	102	92	102	102	102
EX63	6	3	105	92	105	105	105
EX64	6	4	151	104	151	151	151
EX71	7	1	146	94	134	133	134
EX72	7	2	86	80	86	87	86

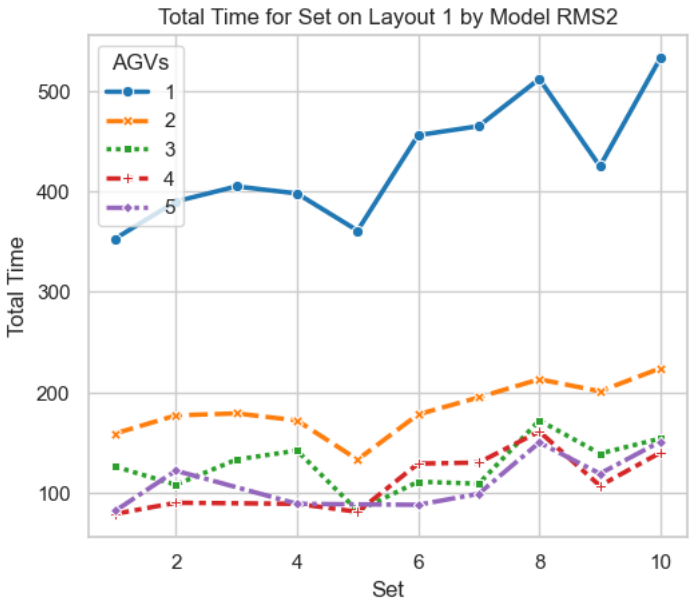
EX73	7	3	93	80	93	95	93
EX74	7	4	169	114	161	161	161
EX81	8	1	167	154	167	167	167
EX82	8	2	155	146	155	155	155
EX83	8	3	155	146	155	155	155
EX84	8	4	178	160	178	178	178
EX91	9	1	127	99	127	127	127
EX92	9	2	106	95	106	106	106
EX93	9	3	107	95	107	107	107
EX94	9	4	149	105	149	149	149
EX101	10	1	153	130	153	153	153
EX102	10	2	139	118	139	139	139
EX103	10	3	139	118	139	139	139
EX104	10	4	183	134	183	183	183

Note. Adapted from "A hybrid particle swarm optimization and simulated annealing algorithm for the job shop scheduling problem with transport resources" by Fontes et al. (2023), *European Journal of Operational Research, 306*(3), 1155. https://doi.org/10.1016/j.ejor.2022.09.006.

The following images represent the possible combinations in which the models were trained and the obtained makespan for each one of the scenarios:

Figure 2.

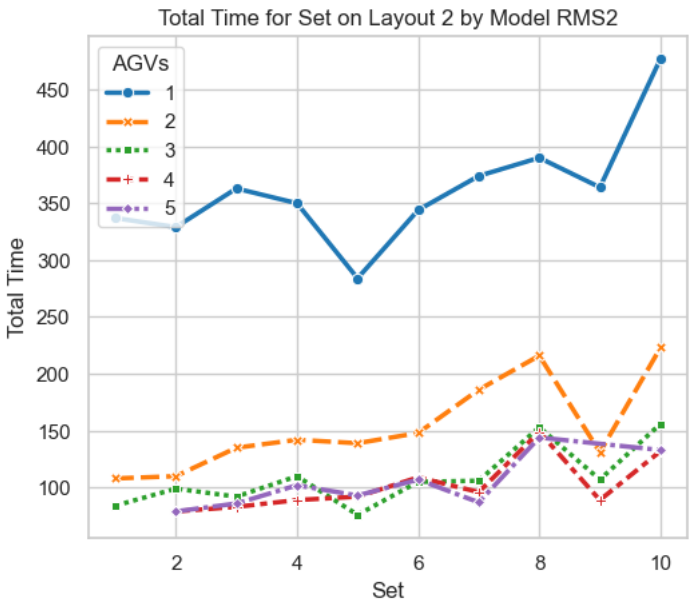
Makespan given AGVs ranges for each layout and its sets.



Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 1.

Figure 3.

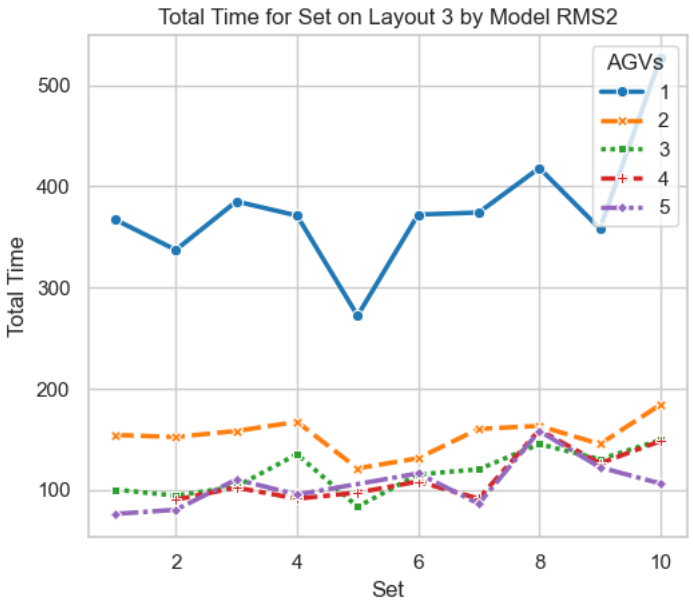
Makespan given AGVs ranges for each layout and its sets.



Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 2.

Figure 4.

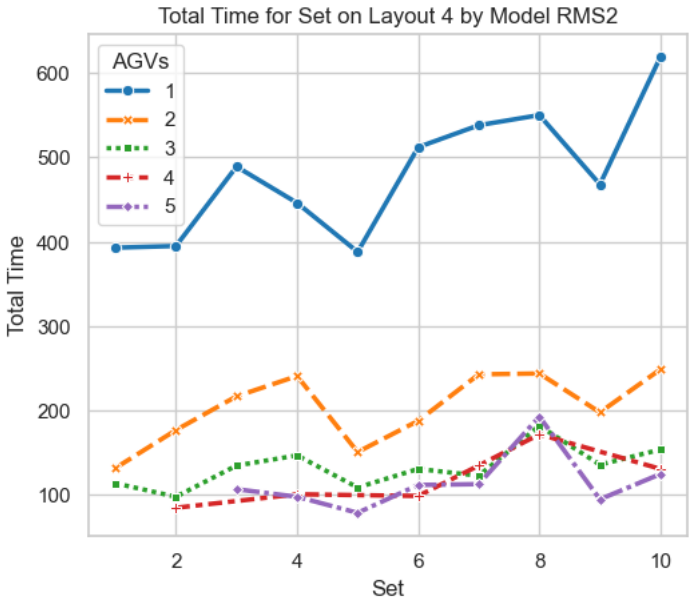
Makespan given AGVs ranges for each layout and its sets.



Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 3.

Figure 5.

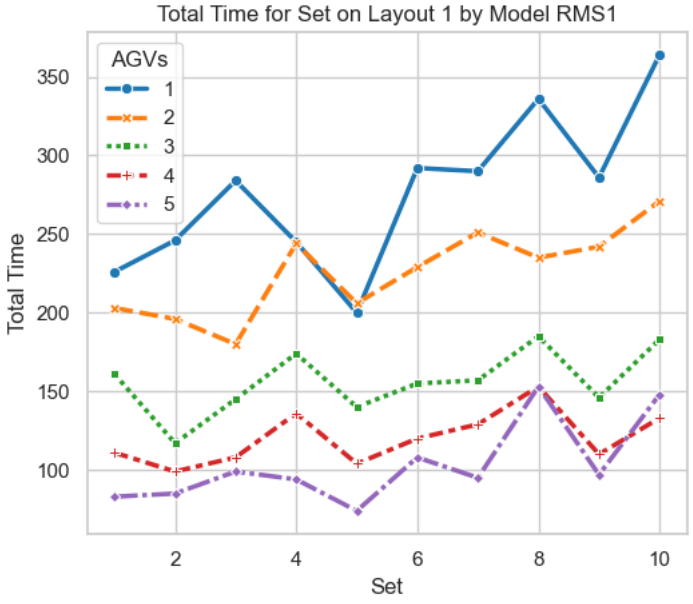
Makespan given AGVs ranges for each layout and its sets.



Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 4.

Figure 6.

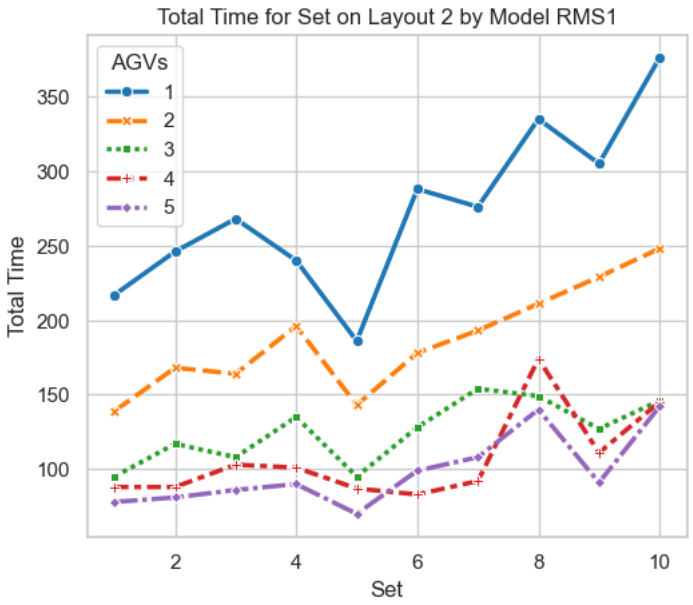
Makespan given AGVs ranges for each layout and its sets.



Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 1.

Figure 7.

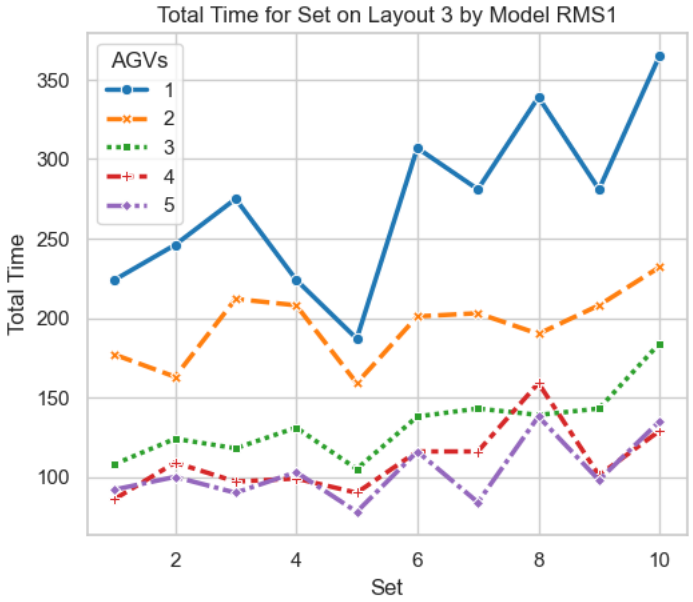
Makespan given AGVs ranges for each layout and its sets.



Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 2.

Figure 8.

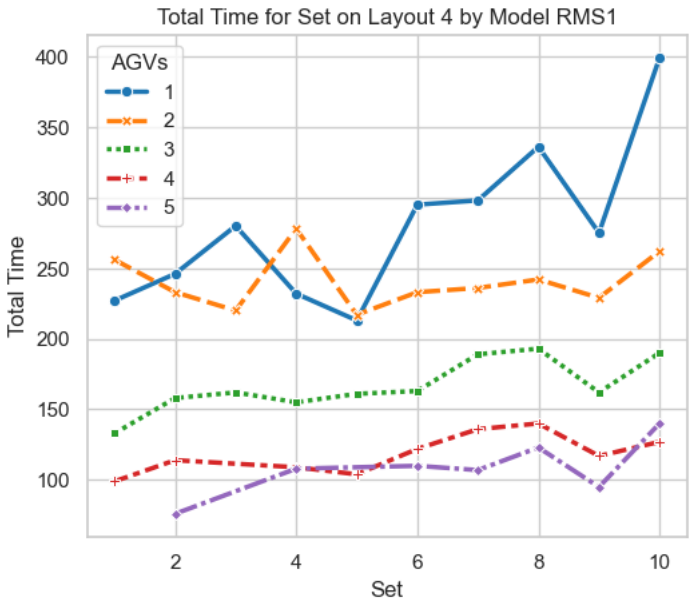
Makespan given AGVs ranges for each layout and its sets.



Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 3.

Figure 9.

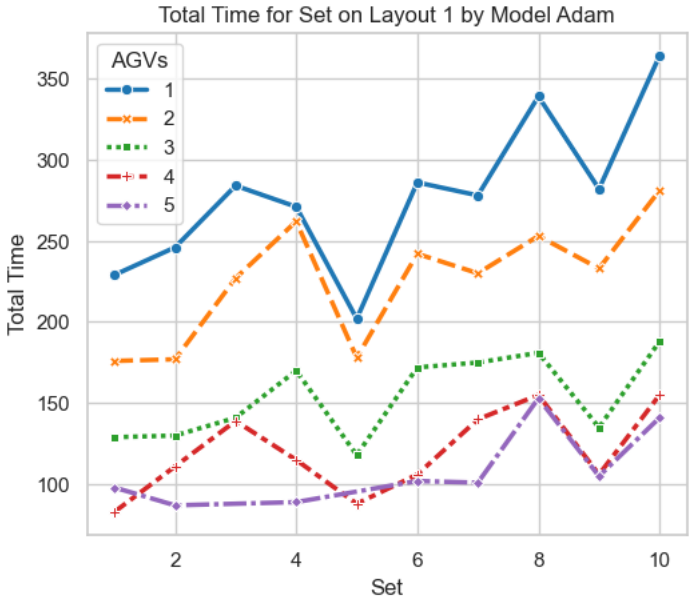
Makespan given AGVs ranges for each layout and its sets.



Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 4.

Figure 10.

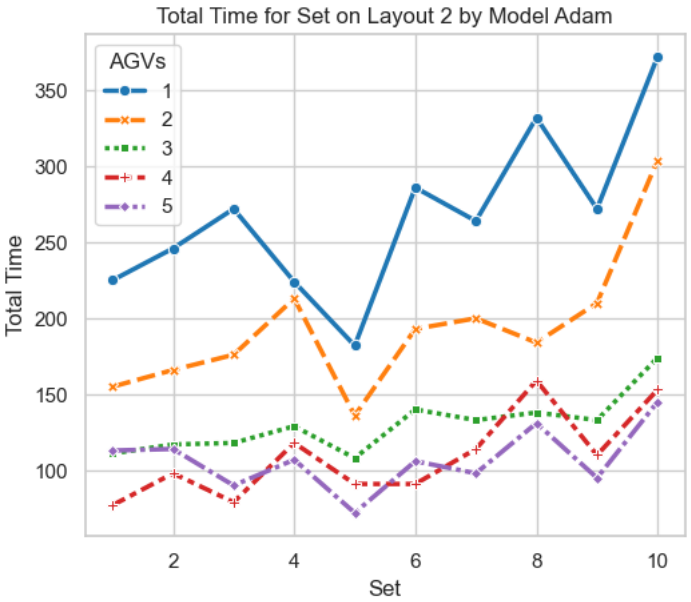
Makespan given AGVs ranges for each layout and its sets.



Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 1.

Figure 11.

Makespan given AGVs ranges for each layout and its sets.



Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 2.

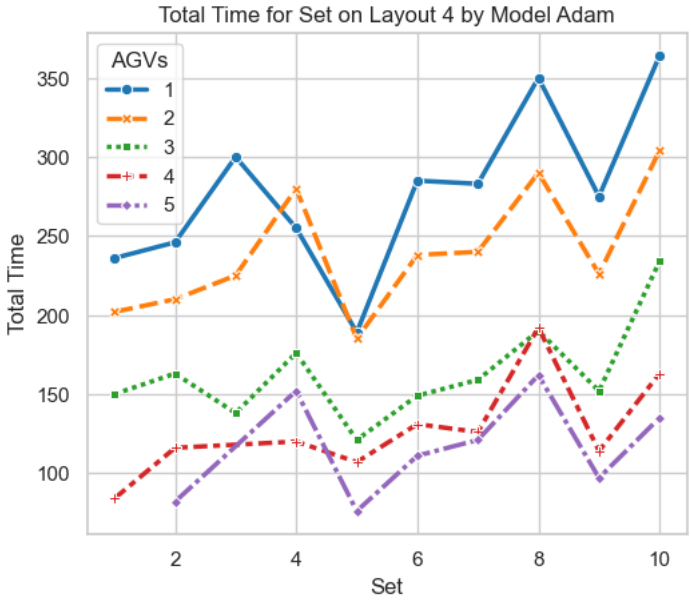
Figure 12.

Makespan given AGVs ranges for each layout and its sets.

Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 3.

Figure 13.

Makespan given AGVs ranges for each layout and its sets.



Description: This graph shows the evolution of total time for each set based on the amount of AGVs on layout 4.

A common issue is that the availability of AGVs or other transport resources alone isn't enough to run the model efficiently. Increasing model robustness led to longer execution times: 20 hours for the first model, 26 for the second, and 46 for the final version under 200 scenarios. Per scenario, times rose from 6 to 8 to 14 minutes. Using multiple GPUs or faster cores can improve these metrics. Additional calculations will assess diverse error metrics, and a simplified Lower Bound will facilitate robust benchmark analysis where AGVs match the number of simultaneous jobs.

8. Analysis

Once the results are obtained, the most important part is to evaluate the performance of the algorithm employed to obtain the results. Therefore, the mean squared error (MSE) as an error measurement performs a vital part in the improvements and measurement of the results, the same as the difference obtained against other solutions and MILP Models implemented.

In this model, Spearman's rank correlation coefficient was chosen to analyze the relationship between the number of jobs and the percentage difference in makespan obtained

in each configuration. Spearman's coefficient is particularly useful here because it measures the strength and direction of a monotonic relationship between two variables, without assuming a linear association or requiring normally distributed data, given that each set has a different number of jobs and processing times. This makes it well-suited for our dataset, where an increase in the number of jobs may correlate with changes in makespan differences in a way that is not strictly linear but potentially consistently increasing or decreasing. Limited resources like AGVs contribute to this difference when the number of jobs exceeds the available AGVs, making scenario evaluation difficult. The table below shows Spearman's correlation coefficient values, and statistical measures against the optimal time to be reached when all resources are available for four scenarios:

- Scenario 1: Second model with RMS optimizer
- Scenario 2: First model with Adam optimizer
- Scenario 3: Second model with RMS optimizer, excluding set 8 from all layouts
- Scenario 4: First model with Adam optimizer, excluding set 8 from all layouts
- Scenario 5: Third model with RMS optimizer
- Scenario 6: Third model with RMS optimizer, excluding set 8 from all layouts

Table 3.

Analysis of deployment against optimal calculated

Scenario	Max Difference	Min Difference	Mean Difference	Spearman's Correlation Coefficient	P-value
Scenario 1	115.38%	1.06%	36.42%	0.6202	0.000019663
Scenario 2	127.69%	0.00%	39.07%	0.6977	0.000000561
Scenario 3	96.08%	1.06%	31.74%	0.6165	0.000048505
Scenario 4	92.16%	0.00%	32.18%	0.7097	0.000001258
Scenario 5	123.77%	0.00%	36.42%	0.6145	0.000014992
Scenario 6	67.31%	0.00%	29.01%	0.6093	0.000049054

The formula for Spearman's rank correlation, denoted as ρ, is based on ranking each variable and computing the correlation of these ranks. It is calculated as:

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \qquad (26)$$

Where d_i represents the difference between the ranks of each pair of observations, and n is the total number of observations. By ranking both variables, Spearman's ρ is less sensitive to outliers than Pearson's correlation coefficient, making it more robust for data that may have irregular distributions or outlying values. The value of ρ ranges from -1 to 1. A ρ value close to 1 indicates a strong positive monotonic relationship (as one variable increases, the other also increases), while a value close to -1 suggests a strong negative monotonic relationship. A ρ near zero implies little to no monotonic association. Thus, Spearman's coefficient enables us to interpret whether an increase in job numbers is associated with an increase or decrease in the makespan difference percentage, providing insight into how scaling the job count may affect the model's deviation from optimal results. In this case, it indicates that there is a moderate to strong positive relationship for both variables (number of jobs and difference obtained from the optimal time with no resource limitation).

Once the difference against the LB proposed is calculated and its behavior has been already identified, indicating a positive relationship between the number of jobs and the difference obtained in percentage. The MSE must be evaluated on the different instances to analyze whether the improvement was effectively implemented. As mentioned above, Table 1 shows the Exit Times (ET) obtained for different types of models in a 2 AGV scenario. The following table resumes the error measurement and its impact on the performance:

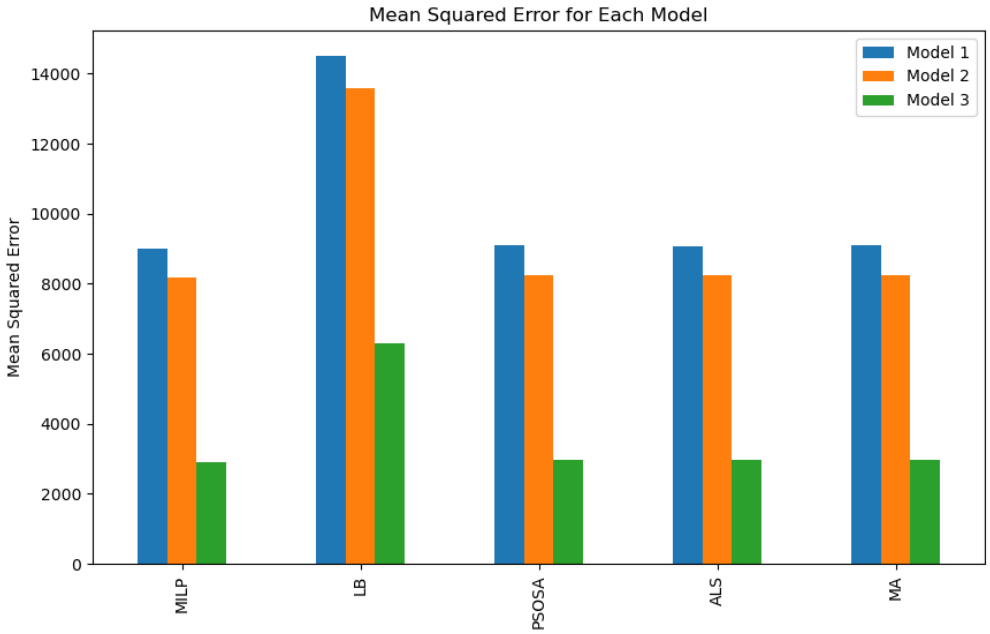
Table 4.

MSE for 2 AGVs instances compared on benchmark

MSE	Model 1	Model 2	Model 3
MILP	9013,425	8163,45	2903,525
LB	14506,475	13596,05	6295,775
PSOSA	9097,425	8258,45	2967,725
ALS	9080,35	8243,125	2956,8
MA	9084,925	8247,75	2964,225

Figure 14.

MSE across benchmark models with the final version



The Mean Squared Error (MSE) plays a critical role in model evaluation, especially within neural networks. As shown in the previous table, MSE values offer a precise and quantitative assessment of a model's accuracy. The following formula indicates how this error metric is considered:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad (27)$$

Where n refers to the number of data points, y_i is the actual point obtained on the neural network developed for this case, and $\hat{y}_i$ is the predicted or estimated value of the i-th data point. For this instance, the data point to be reached based on different algorithms.

A lower MSE signifies that the predicted values are closely aligned with the actual values, reflecting better model performance. There are several benefits to using MSE as an error metric. Firstly, it disproportionately penalizes larger errors due to the squaring of individual errors, prompting the model to minimize substantial deviations and thereby enhance reliability. Secondly, MSE is differentiable, which is vital for optimization algorithms employed in training neural networks. It provides a smooth gradient that aids effective backpropagation and convergence to optimal solutions.

Furthermore, evaluating neural networks with MSE is essential for comparing different models. Employing consistent and standardized error metrics like MSE allows for fair comparisons across various models and datasets. In operations research and statistics, this comparison is crucial for selecting the most suitable model for a specific task. It enables researchers and practitioners to objectively assess performance enhancements and make data-driven decisions. Table 4 illustrates the MSE for different models, demonstrating significant improvements in model evolution. This progress highlights the importance of iterative evaluation and refinement in achieving top-tier model performance. Continuous assessment with metrics like MSE ensures that each iteration improves upon previous versions, reducing errors and boosting accuracy.

In summary, MSE is indispensable in the statistical evaluation of neural networks. It offers a robust framework for minimizing prediction errors, optimizing model training, and facilitating objective comparisons. Focusing on MSE ensures that models are not only accurate but also reliable and effective in real-world applications. These principles are demonstrated by the results, which show a reduction in mean difference—a critical factor in cost-impact decisions for practical applications.

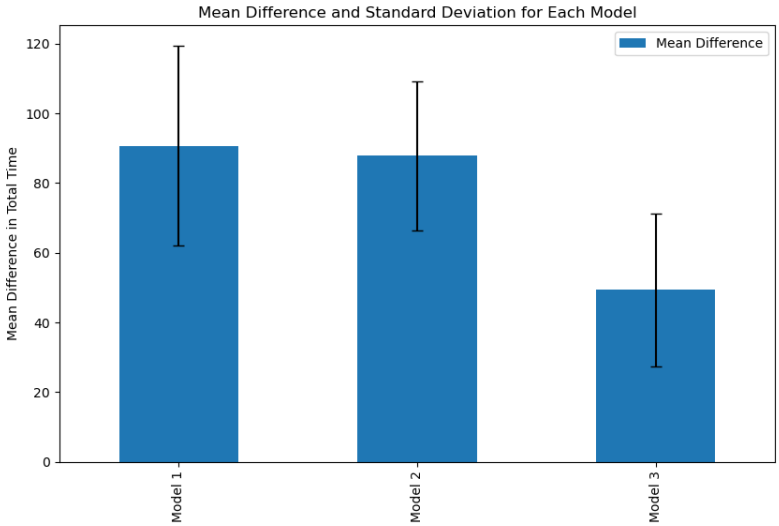
Table 5.

Performance Metrics of Different Models against MILP Makespan

Model	Mean Difference	Standard Deviation
Model 1	90,625	28,654
Model 2	87,850	21,384
Model 3	49,325	21,969

Figure 15.

Performance Metrics of Different Models against MILP Makespan



As detailed in the Results section, the optimal Gap calculated based on (23) is determined for each instance. The current model shows significant improvement, highlighted by substantial reductions in the relevant metrics. This enhancement is attributed to the increase in the number of episodes and the expansion of the buffer, which facilitates learning from past experiences and developing effective policies to address the problem. Although the average gap continues to decrease, it does not reach the optimal solution due to computational time constraints associated with broader data sets. There is a trade-off between optimality, generalization, and computational time; improving one of these aspects impacts the others due to their interdependence. If greater optimality is sought, computational times will rise, and generalization might be affected. Similarly, enhancing generalization could lead to longer computation times and suboptimal optimization, potentially improving but not

guaranteeing a perfect result. Higher computational times support the exploration and exploitation of the other two aspects, contingent on having appropriate resources.

Table 6.

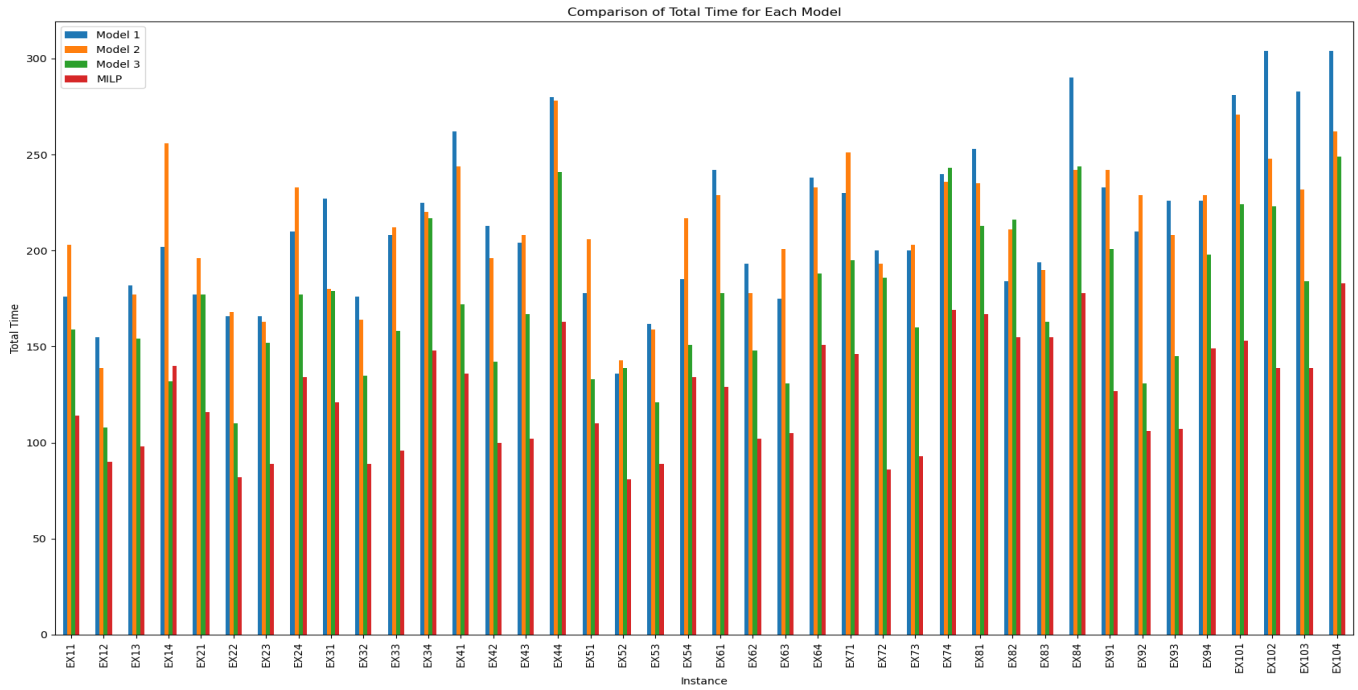
Optimal Gap Metrics

Model	Mean Optimal Gap	Standard Deviation
Model 1	76,48%	0,27189
Model 2	74,78%	0,24720
Model 3	41,63%	0,21275

The improvement in results was attributed to dynamic hyperparameter tuning and optimized data processing techniques within the neural network's tensor, which included modifications that significantly enhanced performance metrics. The alterations in hyperparameters led to the discovery of more efficient scheduling policies, consequently reducing the makespan and increasing the stability of training through larger batch sizes and replay buffers. Moreover, adjusting the epsilon decay rate facilitated the agent's convergence to superior policies by ensuring sustained exploration. However, these advancements necessitated greater computational resources, as larger batch sizes and replay buffers escalated memory consumption, posing a challenge to systems with limited RAM. Additionally, the networks' complexity and the extended training duration demanded meticulous resource management and patience.

Figure 16.

Makespan obtained for 2 AGVs comparing models vs MILP.



Moreover, evaluating finer-tuning options for each instance and scenario would enhance the performance metrics and accuracy of the model, given the nature of a MILP model and every optimization problem. The logic behind a MILP model, which serves as the primary benchmark due to its hybrid approach, is to assess all possible combinations to identify the optimal result. Dynamically modifying hyperparameters can help escape local optima and identify global optima, thereby improving performance against benchmark instances. Performance improvements should be proportionately increased across all scenarios rather than focusing solely on the evaluated ones, considering this could result in higher computational requirements such as increased GPU or CPU consumption and memory usage. By dynamically and randomly evaluating the scenarios, some results exhibited significant improvement. However, in extreme situations where resources are less critical, the model's behavior varies depending on the encoding and decoding methods used. Consequently, it could be adjusted to achieve consistent outcomes and allow for ongoing modifications.

One notable benefit of training these models is their capacity to make predictions for similar scenarios without starting from scratch, as well as their ability to personalize or generalize based on the intended use. Additionally, the model can benefit from Transfer Learning (TL), which facilitates generalization according to the desired application. Evaluating results using different optimizers is essential due to their differing characteristics. While both Adam and RMSProp are designed to adjust learning rates, Adam uses momentum concepts, whereas RMSProp normalizes gradients using their root mean square (Ruder, 2016). This difference leads to varied performance depending on the specific application and data structure. Therefore, dynamic fine-tuning and model selection are crucial to minimize variability, enhance accuracy, and continually measure error metrics for ongoing improvement.

9. Conclusions

This thesis presents a novel approach to the Job Shop Scheduling Problem (JSSP) with transport resources, using a Deep Q-Network (DQN)-based methodology that integrates complex machine scheduling and Automated Guided Vehicle (AGV) coordination within a dynamic reinforcement learning framework. By simulating realistic conditions for job scheduling and transportation within a flexible production environment, this model significantly advances current research in operations management and optimization.

The development process included multiple iterations of the DQN model to enhance robustness and computational efficiency. The model architecture, incorporating prioritized experience replay and a Double DQN framework, enables it to learn high-quality scheduling policies effectively. By balancing exploration and exploitation through an adaptive epsilon-greedy policy, the model dynamically navigates the large action space of JSSP with transport, identifying optimal job-machine-AGV combinations that minimize makespan across various job sets and layouts. Key methodological steps, such as systematic hyperparameter tuning

and prioritized experience replay, allowed the model to handle high-dimensional state spaces and complex scheduling scenarios. This iterative approach—starting from initial neural network structures and progressing through refined architectures—ensures that the model is computationally efficient, scalable, and adaptable to real-world manufacturing environments with limited computational resources, like an 8 GB RAM MacBook M1.

The model was benchmarked against traditional MILP models, demonstrating its capability to approximate optimal scheduling solutions with a smaller optimality gap in shorter times, particularly in configurations with constrained transport resources. Performance metrics, including mean squared error (MSE) and optimality gap, indicate that the DQN model not only improves scheduling efficiency but also achieves results that align with established optimization benchmarks. Moreover, Spearman's correlation analysis illustrates the model's reliability across varying job counts, reinforcing its adaptability in scaling scenarios.

Further refinement could involve extending the training on additional complex scenarios or leveraging Transfer Learning (TL) to apply the model across different job shop environments, ensuring broader applicability in real-world settings. Enhancing computational capacity, such as through distributed systems or GPU utilization, may also yield further improvements in model performance, particularly for larger datasets and more complex scheduling configurations. This work lays a strong foundation for integrating machine learning in production scheduling, with implications for practitioners aiming to streamline operations through advanced scheduling models that incorporate transport resources. The findings encourage ongoing exploration of reinforcement learning-based scheduling and hybrid optimization methods to address the growing demands of agile manufacturing environments through implementing fine-tuning policies and scenarios and measuring the errors to outperform better.

10. Future Work

Potential areas for further research include leveraging cloud computing resources to overcome hardware limitations by utilizing parallelization. This can be achieved by distributing computations across multiple servers or using high-performance cloud-based GPUs, which would enable the training of more complex models or enhance the efficiency of existing ones.

Broadening the model to encompass multi-agent systems is another promising direction. This would involve simulating multiple agents (such as various AGVs or scheduling managers) coordinating within an environment, potentially resulting in more robust and flexible scheduling solutions. Implementing this approach could incorporate algorithms designed for multi-agent reinforcement learning, considering the interactions and shared objectives of multiple agents.

Validating the developed methodology in real-world industrial scenarios using actual data would demonstrate its practical utility. Collaborating with industry partners to access real scheduling data and constraints could reveal additional insights and challenges that are not present in simulated environments. Such real-world applications would also facilitate evaluating the model's scalability and adaptability to different industrial contexts, providing valuable feedback for further refinement.

Furthermore, integrating the proposed environment with other constraints unrelated to scheduling, such as predictive maintenance, battery limitations, and dynamic routing without fixed transport times, but with real-time decision-making by an AGV, could offer a more comprehensive scenario (Dehnavi-Arani et al., 2019). These aspects are addressed in the literature but are not yet unified.

11. References

Abderrahim, A. B. (2023). A multi-agent system simulation-based approach for collision avoidance in integrated job-shop scheduling problem with transportation tasks. *Journal of Manufacturing Systems, 68*, 209–226. https://doi.org/10.1016/j.jmsy.2023.03.011

Alabajee, M., Fadhil, A., & Alsarraj, R. (2020). Job shop scheduling problem: Literature review. *Tikrit Journal of Pure Science*, *25*(4), 91–100. https://doi.org/10.25130/tjps.v25i4.277

Amirteimoori, A., Tirkolaee, E. B., Simic, V., & Weber, G.-W. (2023). A parallel heuristic for hybrid job shop scheduling problem considering conflict-free AGV routing. *Swarm and Evolutionary Computation, 79*, 101312**.** https://doi.org/10.1016/j.swevo.2023.101312

Amjad, M., Ahmad, A., Rehmani, M. H., & Umer, T. (2018). A review of EVs charging: From the perspective of energy optimization, optimization approaches, and charging techniques. *Transportation Research Part D: Transport and Environment, 62*, 386-417. https://doi.org/10.1016/j.trd.2018.03.006

Awad, M., & Abd-Elaziz, H. (2021). A new perspective for solving manufacturing scheduling based problems respecting new data considerations. *Processes*, *9*(10), 1700. https://doi.org/10.3390/pr9101700

Azadeh, K., Roshanaei, V., Hatami, S., & Maleki, A. (2020). A MILP model for energy-efficient job shop scheduling problem with transport resources. *INRIA HAL* Science. https://inria.hal.science/hal-02992767

Azzouz, A., Ennigrou, M., & Ben Said, L. (2017). A self-adaptive hybrid algorithm for solving flexible job-shop problem with sequence dependent setup time. *Procedia Computer Science, 112*, 457-466. https://doi.org/10.1016/j.procs.2017.08.023

Ba, L., Li, Y., & Yang, M. (2016). Modelling and simulation of a multi-resource flexible job-shop scheduling. *International Journal of Simulation Modelling*, *15*(1), 157–169. https://doi.org/10.2507/ijsimm15(1)co3

Bilge, U., & Ulusoy, G. (1995). Time window approach to simultaneous scheduling machines and material handling system in an FMS. *Operations Research*, *43*(6), 1058–1071. https://doi.org/10.1287/opre.43.6.1058

Bozzi, A. (2023). Dynamic MPC-based scheduling in a smart manufacturing system problem. *IEEE Access*, *11*, 141987–141996. https://doi.org/10.1109/access.2023.3341504

Canese, L., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M., & Spanò, S. (2021). Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences, 11*(11), 4948. https://doi.org/10.3390/app11114948

Chang, J., Yu, D., Hu, Y., He, W., & Yu, H. (2022). Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival. *Processes*, *10*(4), 760. https://doi.org/10.3390/pr10040760

Chen, Z., Zhang, L., Wang, X., & Gu, P. (2022). Optimal design of flexible job shop scheduling under resource preemption based on deep reinforcement learning. *Complex Systems Modeling and Simulation*, *2*(2), 174–185. https://doi.org/10.23919/csms.2022.0007

Chen, T., Bu, S., Li, X., Kang, J., Yu, F., & Han, Z. (2022). Peer-to-peer energy trading and energy conversion in interconnected multi-energy microgrids using multi-agent deep reinforcement learning. *IEEE Transactions on Smart Grid*, *13*(1), 715–727. https://doi.org/10.1109/tsg.2021.3124465

Dehnavi-Arani, S., Sabaghian, A., & Fazli, M. (2019). A job shop scheduling and location of battery charging storage for the automated guided vehicles (AGVs). *Journal*

of Optimization in Industrial Engineering, 12(2), 121-129. https://doi.org/10.22094/JOIE.2018.543203.1511

Fan, H., & Su, R. (2022). Mathematical modelling and heuristic approaches to job-shop scheduling problem with conveyor-based continuous flow transporters. *Computers & Operations Research, 148*, 105998. https://doi.org/10.1016/j.cor.2022.105998

Fontes, D. B. M. M., Homayouni, S. M., & Gonçalves, J. F. (2023). A hybrid particle swarm optimization and simulated annealing algorithm for the job shop scheduling problem with transport resources. *European Journal of Operational Research*, *306*(3), 1140–1157. https://doi.org/10.1016/j.ejor.2022.09.006

Fontes, D. B. M. M., Homayouni, S. M., & Resende, M. G. C. (2022). Job-shop scheduling—joint consideration of production, transport, and storage/retrieval systems. *Journal of Combinatorial Optimization*, *44*(2), 1284–1322. https://doi.org/10.1007/s10878-022-00885-8

Han, B., & Yang, J. (2020). Research on adaptive job shop scheduling problems based on dueling double DQN. *IEEE Access*, *8*, 186474–186495. https://doi.org/10.1109/access.2020.3029868

Han, B., & Yang, J. (2021). A deep reinforcement learning based solution for flexible job shop scheduling problem. *International Journal of Simulation Modelling*, *20*(2), 375–386. https://doi.org/10.2507/ijsimm20-2-co7

Homayouni, S. M., & Fontes, D. B. M. M. (2021). A MILP model for energy-efficient job shop scheduling problem and transport resources. *Advances in Production Management Systems: Artificial Intelligence for Sustainable and Resilient Production Systems. IFIP Advances in Information and Communication Technology, 630, 378-387.* https://doi.org/10.1007/978-3-030-85874-2_1

Jungbluth, S., Gafur, N., Popper, J., Yfantis, V., & Ruskowski, M. (2022). Reinforcement learning-based scheduling of a job-shop process with distributedly controlled robotic manipulators for transport operations. *IFAC PapersOnLine, 55*(2), 156–162. https://doi.org/10.1016/j.ifacol.2022.04.186

Kim, S., Neale, V., Chowdhary, G., & Tran, H. (2022). Disentangling successor features for coordination in multi-agent reinforcement learning. https://doi.org/10.48550/arxiv.2202.07741

Kim, S., Neale, V., Chowdhary, G., & Tran, H. (2022). Disentangling successor features for coordination in multi-agent reinforcement learning. *arXiv preprint.* https://doi.org/10.48550/arxiv.2202.07741

Kirilov, L., & Guliashki, V. (2017). An algorithm for generating a dispersed population of feasible schedules for flexible job shop problems. *Information Technologies and Control*, *15*(3), 16–19. https://doi.org/10.1515/itc-2017-0029

Kumar, R., Tiwari, M., & Shankar, R. (2003). Scheduling of flexible manufacturing systems: An ant colony optimization approach. *Proceedings of the Institution of Mechanical Engineers Part B: Journal of Engineering Manufacture*, *217*(10), 1443–1453. https://doi.org/10.1243/095440503322617216

Latthawanichphan, J., Songserm, W., & Wuttipornpun, T. (2019). An instance generator for scheduling problems featuring options for unequal stages and unequal parallel machines. *International Journal of Technology and Engineering Studies*, *5*(4), 106–112. https://doi.org/10.20469/ijtes.5.10001-4

Li, H., Duan, J., & Zhang, Q. (2020). Multi-objective integrated scheduling optimization of semi-combined marine crankshaft structure production workshop for green manufacturing. *Transactions of the Institute of Measurement and Control*, *43*(3), 579–596. https://doi.org/10.1177/0142331220945917

Li, W., Han, D., Gao, L., Li, X., & Li, Y. (2022). Integrated production and transportation scheduling method in hybrid flow shop. *Chinese Journal of Mechanical Engineering*, *35*(1). https://doi.org/10.1186/s10033-022-00683-7

Li, Y., Chen, X., An, Y., Zhao, Z., Cao, H., & Jiang, J. (2023). Integrating machine layout, transporter allocation and worker assignment into job-shop scheduling solved by an improved non-dominated sorting genetic algorithm. *Computers & Industrial Engineering, 179*, 109169. https://doi.org/10.1016/j.cie.2023.109169

Lv, S. (2024). GABB: The plan-based job scheduling optimized by genetic algorithm for HPC systems with shared burst buffers., 19, 109. https://doi.org/10.1117/12.3034965

Momenikorbekandi, A., & Abbod, M. F. (2023). A novel metaheuristic hybrid parthenogenetic algorithm for job shop scheduling problems: Applying an optimization model. *IEEE Access, 11*, 56027-56045. https://doi.org/10.1109/ACCESS.2023.3278372

Meng, L., Zhang, B., Gao, K., & Duan, P. (2023). An MILP model for energy-conscious flexible job shop problem with transportation and sequence-dependent setup times. *Sustainability*, *15*(1), 776. https://doi.org/10.3390/su15010776

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

Moin, N., Sin, O., & Omar, M. (2015). Hybrid genetic algorithm with multiparents crossover for job shop scheduling problems. *Mathematical Problems in Engineering*, *2015*, 1–12. https://doi.org/10.1155/2015/210680

Muthiah, A., Rajkumar, A., & Rajkumar, R. (2016). Hybridization of Artificial Bee Colony algorithm with Particle Swarm Optimization algorithm for flexible Job Shop Scheduling. https://doi.org/10.1109/iceets.2016.7583875

Parveen, S. and Ullah, H. (2011). Review on job-shop and flow-shop scheduling using. Journal of Mechanical Engineering, 41(2), 130-146. https://doi.org/10.3329/jme.v41i2.7508

Ren, J., Ye, C., & Li, Y. (2020). A two-stage optimization algorithm for multi-objective job-shop scheduling problem considering job transport. *Journal Européen des Systèmes Automatisés, 53*(6), 915-924. https://doi.org/10.18280/jesa.530617

Ramasubbareddy, S., Swetha, E., Luhach, A., & Srinivas, T. (2021). A multi-objective genetic algorithm-based resource scheduling in mobile cloud computing. *International Journal of Cognitive Informatics and Natural Intelligence*, *15*(3), 58–73. https://doi.org/10.4018/ijcini.20210701.oa5

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*. https://doi.org/10.48550/arXiv.1609.04747

Song, L., Liu, C., & Shi, H. (2022). Discrete particle swarm algorithm with Q-learning for solving flexible job shop scheduling problem with parallel batch processing machine. *Journal of Physics Conference Series*, *2303*(1), 012022. https://doi.org/10.1088/1742-6596/2303/1/012022

Song, L., Liu, C., Shi, H., & Zhu, J. (2022). An improved immune genetic algorithm for solving the flexible job shop scheduling problem with batch processing. *Wireless Communications and Mobile Computing*, 2022, 1–17. https://doi.org/10.1155/2022/2856056

Tassel, P., Kovács, B., Gebser, M., Schekotihin, K., Kohlenbrein, W., & Schrott-Kostwein, P. (2022). Reinforcement learning of dispatching strategies for large-scale industrial scheduling. *Proceedings of the International Conference on Automated Planning and Scheduling*, 32, 638–646. https://doi.org/10.1609/icaps.v32i1.19852

Wang, Y., Liu, H., Zheng, W., Xia, Y., Li, Y., Chen, P., … & Xie, H. (2019). Multi-objective workflow scheduling with Deep-Q-network-based multi-agent reinforcement learning. *IEEE Access*, *7*, 39974–39982. https://doi.org/10.1109/access.2019.2902846

Wu, S., Wang, T., Li, C., & Zhang, C. (2021). Containerized distributed value-based multi-agent reinforcement learning. https://doi.org/10.48550/arxiv.2110.08169

Xu, Z., Zhang, B., Li, D., Zhang, Z., Zhou, G., Chen, H., … & Fan, G. (2022). Consensus learning for cooperative multi-agent reinforcement learning. https://doi.org/10.48550/arxiv.2206.02583

Yao, Y., Liu, Q., Li, X., & Gao, L. (2023). A novel MILP model for job shop scheduling problem with mobile robots. *Robotics and Computer-Integrated Manufacturing, 81*, 102506. https://doi.org/10.1016/j.rcim.2022.102506

Yu, H., Gao, Y., Wang, L., & Meng, J. (2020). A hybrid particle swarm optimization algorithm enhanced with nonlinear inertial weight and Gaussian mutation for job shop scheduling problems. *Mathematics*, *8*(8), 1355. https://doi.org/10.3390/math8081355

Zambrano-Rey, G. M., González-Neira, E. M., Forero-Ortiz, G. F., Ocampo-Monsalve, M. J., & Rivera-Torres, A. (2023). Minimizing the expected maximum lateness for a job shop subject to stochastic machine breakdowns. *Annals of Operations Research*. https://doi.org/10.1007/s10479-023-05592-z

Zeng, R., & Wang, Y. (2018). A chaotic simulated annealing and particle swarm improved artificial immune algorithm for flexible job shop scheduling problem. *Eurasip Journal on Wireless Communications and Networking*, 2018(1). https://doi.org/10.1186/s13638-018-1109-2

Zhao, Y., Wang, Y., Tan, Y., Zhang, J., & Huang, Y. (2021). Dynamic jobshop scheduling algorithm based on deep Q network. *IEEE Access*, *9*, 122995–123011. https://doi.org/10.1109/access.2021.3110242

Zhao, Y., & Zhang, H. (2021). Application of machine learning and rule scheduling in a job-shop production control system. *International Journal of Simulation Modelling*, *20*(2), 410–421. https://doi.org/10.2507/ijsimm20-2-co10

Zhou, W. (2024). Integrated scheduling algorithm with dynamic adjustment on machine idle time. https://doi.org/10.21203/rs.3.rs-4302637/v1

Zhou, W., Zhou, P., Yang, D., Cao, W., Tan, Z., & Xie, Z. (2023). Symmetric two-workshop heuristic integrated scheduling algorithm based on process tree cyclic decomposition. *Electronics*, *12*(7), 1553. https://doi.org/10.3390/electronics12071553

Zhou, W., Zhou, P., Zheng, Y., & Xie, Z. (2022). A heuristic integrated scheduling algorithm via processing characteristics of various machines. *Symmetry*, *14*(10), 2150. https://doi.org/10.3390/sym14102150

Zhang, C., & Zheng, R. (2023). Event-driven dynamic job-shop scheduling method with strong process constraints. *Journal of Computing and Electronic Information Management*, *10*(3), 72–79. https://doi.org/10.54097/jceim.v10i3.8705

Zhang, F., Mei, Y., Nguyen, S., Zhang, M., & Tan, K. (2021). Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation*, *25*(4), 651–665. https://doi.org/10.1109/tevc.2021.3065707

Zhang, M., Lü, Y., Hu, Y., Amaitik, N., & Xu, Y. (2022). Dynamic scheduling method for job-shop manufacturing systems by deep reinforcement learning with proximal policy optimization. *Sustainability*, *14*(9), 5177. https://doi.org/10.3390/su14095177

12. Annexes

Code Repository on GitHub with three versions of the model developed:

https://github.com/jpuerto1604/JSSwTransport