# Using Graph Neural Networks to Solve Scheduling Problems

**Student:** Hazar Çakr
**Supervisor:** Prof. Dr. Martin Grunow
**Advisor:** Jan-Niklas Dörr
TUM School of Management
Technical University of Munich
Munich Germany

`hazar.cakir@tum.de`
`jan.doerr@tum.de`

June 24, 2024

**Abstract**

This report presents the development and evaluation of advanced computational models to optimize job scheduling on a production line. The project commenced with the creation of a simulation environment to mirror real-world operations of machines and orders within a production line, aiming to minimize the completion time or maximize the throughput of orders before their due dates. Initially, an automatic algorithm was implemented to establish a baseline for job scheduling efficiency. Subsequently, a deep reinforcement learning system utilizing the Proximal Policy Optimization (PPO) method was developed, which showed promising results. To enhance the model's adaptability to varying initial conditions and to leverage structural data, traditional neural network layers were replaced with graph neural layers, culminating in a graph neural network-based PPO system. Comprehensive benchmarking of these models demonstrated significant improvements in scheduling efficiency, underscoring the potential of graph neural networks in dynamic and flexible problem-solving environments. This report analyzes the performance variations between the models and discusses the implications of these technologies in industrial applications.

## 1 Introduction

Job scheduling is a fundamental problem in operations research and industrial engineering, involving the allocation of jobs to resources at specific times to optimize one or more objectives. The job-shop scheduling problem, a variant of the general job scheduling issue, specifically deals with assigning and sequencing jobs on machines, where each job consists of a sequence of tasks that must be performed in a prescribed order. This problem is notorious for its computational complexity, especially as the size of the operation increases.

Traditionally, job-shop scheduling problems have been approached through various optimization techniques, including linear programming, constraint programming, and heuristic-based methods such as genetic algorithms, simulated annealing, and tabu search. These methods focus on finding feasible solutions in a reasonable time frame rather than guaranteeing the optimal solution due to the NP-hard nature of the problem.

However, with the increasing complexity and dynamism of modern manufacturing systems, these traditional methods often fall short in adaptability and scalability. This gap has prompted the exploration of machine learning techniques, which can potentially provide more flexible and efficient solutions. Machine learning models, particularly those employing reinforcement learning, can iteratively learn from the environment to make decisions that cumulatively lead to an optimal or near-optimal solution without needing explicit programming for every possible scenario.

The integration of machine learning into job scheduling introduces the capability to handle high-dimensional and dynamic data effectively. The advent of deep learning has further expanded these capabilities. Specifically, the use of graph neural networks (GNNs) in this context represents a significant innovation. GNNs are particularly suited for job-shop scheduling as they naturally leverage the relational data structure of jobs and machines. Unlike traditional neural networks, GNNs can handle variable input sizes and structures, allowing for a more flexible setup during training and operation. This flexibility is crucial for adapting to different production configurations and evolving requirements without re-engineering the solution from scratch, offering a substantial advantage over traditional and earlier machine learning approaches.

In this project, we explore the application of both conventional deep learning models and graph neural networks to the job-shop scheduling problem, aiming to highlight the benefits and potential of GNNs in achieving greater adaptability and efficiency in production line optimization.

The research conducted was part of a interdisciplinary project under the supervision of Prof. Dr. Martin Grunow and M.Sc. Jan-Niklas Dörr . It represents a critical component of my curriculum, providing a unique opportunity to apply theoretical knowledge to a complex, real-world problem. This project not only allowed me to deepen my understanding of reinforcement learning but also to utilise my prior experience in graph neural networks in a different area. The process and outcomes of this research are detailed in this report, which serves as the final report for the project.

## 2 Related Work

During the implementation of the project, several papers are investigated which try to handle the Job-Shop Scheduling Problem (JSSP) with the help of the Graph Neural Networks (GNNs). Each project has slightly different problem set-up and different approaches.

In the initial phase of this interdisciplinary project, a detailed analysis and implementation were undertaken based on the methodologies outlined in "A General Automatic Method for Optimal Construction of Matrix Product Operators Using Bipartite Graph Theory" by Park et al. (2021) [1]. The study introduces a framework that utilizes graph neural networks and reinforcement learning to optimize JSSP. Employing Proximal Policy Optimization (PPO), the framework enhances scheduling decisions by learning directly from the structure of JSSP represented as a graph. This method claims to outperform traditional dispatching rules and to have the ability to adapt to new JSSP configurations without further training. The adaptability and effectiveness of this approach provide a valuable reference for the development of our project's GNN-based scheduling system.

Another paper that works on the same problem with similar approach is "Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning" from Zhang et al. (2020) [2]. In this paper, a clear blueprint for the graph embeddings is provided called as "Disjunctive graph", which is a common way to represent the JSSP problems. Even though the general structure is pretty similar, this detailed explanation set a basis to our research to come up with own graph embeddings. Also the reward function in the paper seems more promising and influenced our project.

Another paper on this topic is "Flexible Job-Shop Scheduling via Graph Neural

Network and Deep Reinforcement Learning" by Song et al. (2023) [3]. This paper has a different approach on the graph embeddings which utilisez the resources as different nodes called as "Heterogeneous Graph", which inspires our implementation a lot. Also the code from the paper is public, which provides a practical example to the project.

These foundational texts have guided the initial stages of our exploration, setting a theoretical and practical framework within which our project was positioned. Through the critical examination and application of these sources, our research endeavors to build upon the established knowledge base, addressing gaps and extending the methodology to new contexts.

## 3 Problem Definition

The focus of this report is on addressing Job Shop Scheduling Problems (JSSP), which involve the assignment and sequencing of jobs to machines. Each job in JSSP comprises a series of tasks that need to be completed in a specific order. To tackle this, we have set up a simulated environment that mirrors the complexities of real-world JSSPs. Due to the intricate nature of these problems and the urgency with which scheduling decisions must be made, traditional mathematical solvers are often impractical. Consequently, we have developed a deep reinforcement learning (DRL) model utilizing GNNs designed to learn efficient scheduling of orders and jobs. In this section, we will detail the simulated environment and clarify relevant terminology.

We will represent the scheduling problem using a production line, which includes multiple machines referred to as "Resources." These resources are systematically organized into stages. Each resource is assigned to a specific stage, and the stages are sequentially numbered. Resources within a stage may be preceded or succeeded by resources from adjacent stages, facilitating a structured flow of operations across the production line.

In our model, the items produced on the production line are referred to as "Orders." Each order is associated with a specific "Product" type and has a due date by which it should be completed. The system defines various product types, each with distinct characteristics. Products are composed of "Operations," which must be executed in a sequential order. Each operation occurs at a designated stage within the production line. However, the compatibility between operations and resources varies depending on the product type. For instance, consider "Operation 1" at stage 1; the resources compatible with this operation may differ for "Product

1" compared to "Product 2," even within the same stage.

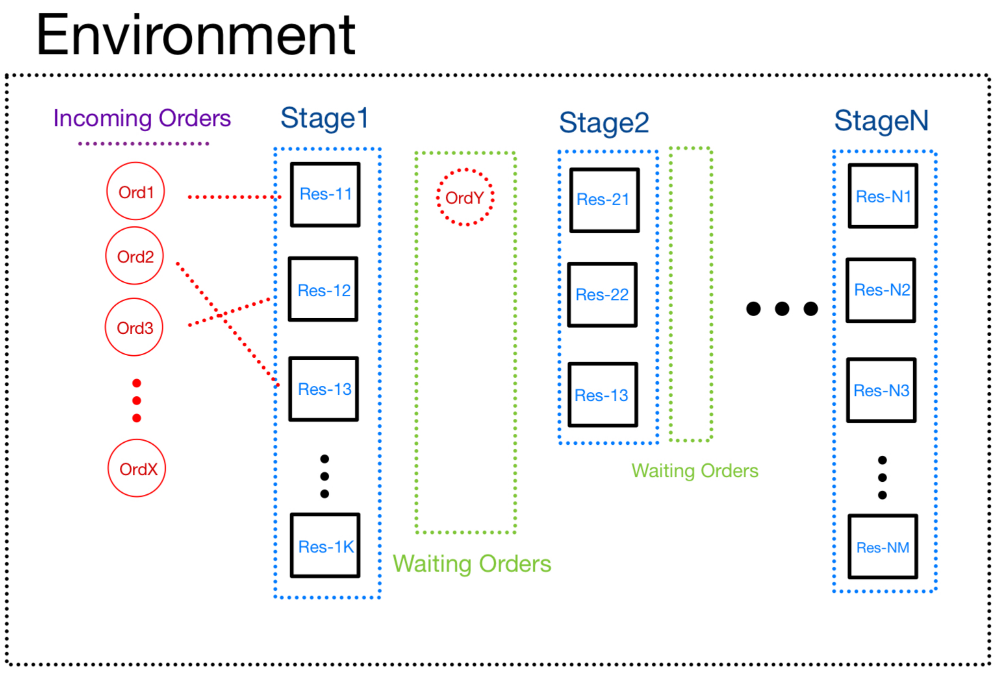


Figure 1: The environment with N stages.

A particular aspect of our scheduling problem involves "Set-up" times. Each product type has associated "Set-up Times" and "Processing Times" when processed on a resource. The processing time refers to the duration required to manufacture a given product type in that resource. In contrast, set-up time is the period needed to prepare the resource for processing a different type of product. If successive products processed on a resource are of the same type, the set-up time is effectively zero, indicating no additional time is required for preparation. However, when a resource switches between different product types, a specific set-up time is incurred to ready the resource for each new product type.

Another dimension of the scheduling challenge is the flexibility in selecting resources within a stage for each order. Each operation in an order offers the possibility of choosing from among compatible resources in a stage. This choice represents a critical juncture in the decision-making process. Without this level of flexibility, the primary decision would merely involve sequencing the "Orders," making the rest of the process relatively deterministic.

Orders may experience waiting periods between stages after being processed by a resource, although they release the resource immediately following the processing time. The primary objective of the Job Shop Scheduling Problem (JSSP) is to optimize the allocation of orders by adhering to the prescribed sequence of operations. During the simulation, two optimization conditions are considered: minimizing the total time span of production across all orders, and reducing tardiness, which measures the delay in production of each order. The main emphasis of this project is on minimizing the total time span, aiming to enhance overall production efficiency.

Please check Figure 1 to detailed illustration of the production environment.

## 4 Methodology

Similar to other deep reinforcement learning (DRL) systems, our project required the creation of a simulation environment that accurately represents the system being optimized. This environment includes essential components such as reset, step, observation space, and action space attributes/methods. As an initial phase, this corresponding environment was implemented, providing a foundation for the subsequent development of the DRL system.

Several approaches exist within deep reinforcement learning; we selected the Proximal Policy Optimization (PPO) model for its proficiency in managing complex, dynamic, and uncertain environments. Rather than utilizing a pre-existing model from a library, we opted to implement the model ourselves. This approach allowed us to seamlessly integrate Graph Neural Network (GNN) layers, enhancing the model's adaptability to our specific requirements.

Our model comprises two sub-models: the GNN-State model, which learns and represents the state of the environment, and the Actor-Critic model, which processes this state to determine the appropriate actions. Both models are trained simultaneously, with the output from the GNN-State model serving as input for the Actor-Critic model. The training of the Actor-Critic model adheres to standard Proximal Policy Optimization (PPO) principles: it employs a policy gradient method for optimization, using clipped probability ratios to ensure that updates do not deviate excessively from the current policy, thus promoting stable and consistent learning. Additionally, PPO alternates between sampling data through interaction with the environment and optimizing a "surrogate" objective function, maximizing efficiency by reusing data across multiple epochs of stochastic gradient ascent.

First, lets discuss the GNN-State model before moving on to the Actor-Critic model. This requires further explanaiton on Graph Neural Networks. GNNs are a class of deep learning models designed to capture the dependence of graphs via message passing between the nodes of graphs. Unlike traditional neural networks, which assume that inputs are independent of each other, GNNs maintain the relational information inherent in the data, making them particularly well-suited for tasks where data can be naturally represented as graphs, such as social networks, chemical compounds, or even the job-shop scheduling problems discussed in this project.

GNNs operate on the principle of message passing, where nodes aggregate information from their neighbors through successive layers. This process can be broadly divided into two steps: aggregation and update. During aggregation, each node collects and combines messages from its neighbors, typically using functions such as sum, mean, or max to ensure the process is permutation invariant. In the update step, nodes then update their own state by applying a transformation, usually via a neural network, to the aggregated information. This methodology allows GNNs to learn complex patterns in the connectivity and properties of graphs. [4]

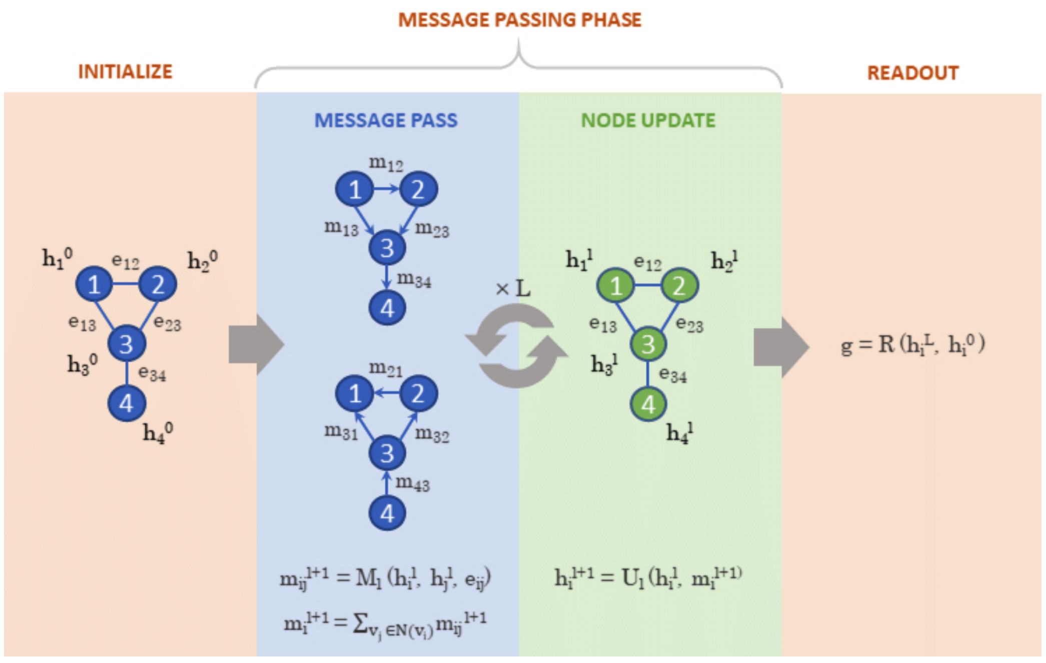

MESSAGE PASSING PHASE
INITIALIZE
READOUT
MESSAGE PASS
NODE UPDATE
× L
$m_{ij}^{l+1} = M_l(h_i^l, h_j^l, e_{ij})$
$m_i^{l+1} = \Sigma_{v_j \in N(v_i)} m_{ij}^{l+1}$
$h_i^{l+1} = U_l(h_i^l, m_i^{l+1})$
$g = R(h_i^L, h_i^0)$

Figure 2: General message-passing procedure for a Graph Neural Network from *Mercado, (2021). [5]*

The structural design of GNNs varies, but most architectures include several layers of these message passing steps to enable deep learning on graphs. Each layer outputs a new set of node features, which encapsulate not only the properties of the nodes but also the topological structure of the graph around them. [6]

In the graph structure of our model, each node symbolizes a resource within our environment, illustrating how the production line is represented. The graph's layout mirrors the structure of the production line, where stages are delineated and nodes between two adjacent stages are fully connected. Dummy nodes are added to the left and right extremes of the graph, linking to the first and last layers of nodes. These connections are bidirectional across the graph, except for the dummy nodes, allowing for two-way information flow. Each node contains a feature vector that holds all relevant information about the resource it represents, as well as details about the current order being processed. This method differs from traditional approaches by adopting a more graphcentric perspective, emphasizing the relationships between resources rather than focusing solely on products or operations. The details of the graph structure can be seen in Figure 3.

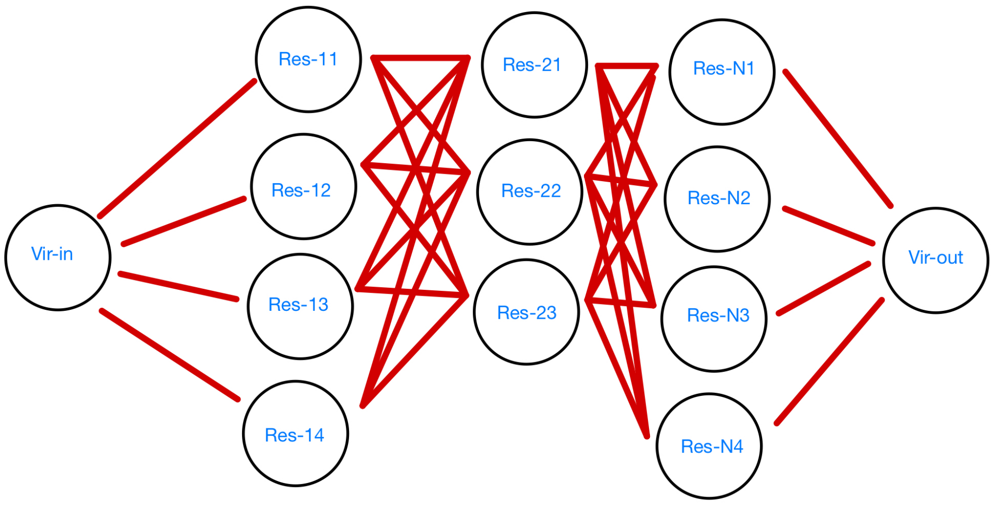


Figure 3: The GNN-State graph structure. Each node corresponds to a resource and they alligned as "Stages" just like environment.

The Actor-Critic model is composed of two separate sub-models, each featuring 3-4 fully connected hidden layers. The dimensions of these hidden layers and the number of layers themselves are hyperparameters that require tuning. During each forward step, an action mask is applied to sift through and identify valid actions. This mask is derived from simulations of the environment and plays a vital role in the decision-making process.

## 5 Results and Evaluations

To evaluate our model, we have devised various setups that each assess different aspects of the problem. A robust method for testing the model and assessing its reliability involves constructing scenarios that increase in complexity. Consequently, we identify two dimensions of complexity: flexibility in resource selection and the incorporation of set-up times, which are explained in the Section 3 - Problem Definition.

Flexibility refers to the ability to choose among different resources within a stage for each order. This significantly complicates the decision-making process by introducing numerous conflicting options. Set-up times represent the duration required to prepare a resource for processing a different type of product. The presence of set-up times complicates the decision of which order to process next. Given their substantial impact on the final outcome, set-up times add an entirely new dimension to the problem.

We employ two distinct assessment processes to evaluate our models: examining the gaps between the optimal solutions for specific samples, and assessing the model's ability to generalize.

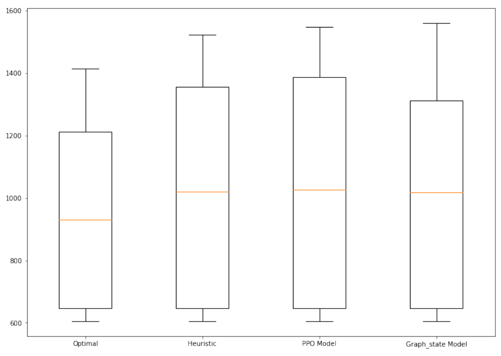
For the first assessment, we generate various instances across different configurations, each reflecting the complexities previously outlined. These instances are tested against both the optimal solution and a heuristic benchmark, allowing us to identify and quantify the gaps between our models performance and these reference solutions. The results are summarized in the table 1 and displayed using box plots in figure 4.

In the second assessment, we focus on one of the configurations where our model shows promise. Here, we generate a lot of random instances and train and test them. Given the high computational cost of calculating optimal solutions, we limit the use of these to the first assessment only. For this second scenario, we rely solely on heuristic benchmarks. The results from this process can be seen in a violin graph in figure 5 to visually represent the performance distribution and deviations.

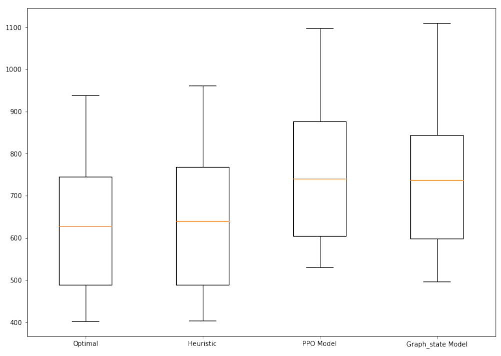
Since we employ a heuristic method as a benchmark, it's vital to describe the heuristic approach used in these tests. This method tackles the problem through a greedy strategy. At each decision point, the heuristic selects the action that has the shortest operation time, which is determined by the combined set-up and processing times of the action. While this straightforward approach may not closely approach the optimal solution, it still vastly outperforms random action selection, demonstrating reasonable effectiveness.

## 5.1 Gap between the optimal solutions

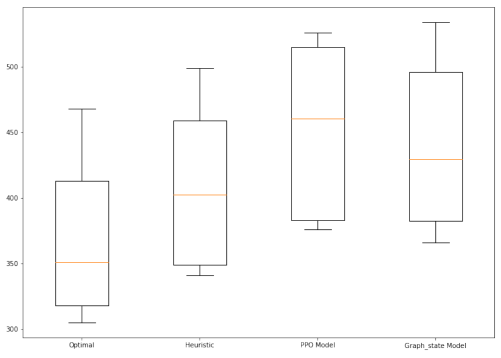
We have developed four distinct cases based on combinations of these factors to examine the discrepancy between the optimal solutions for given samples. The results about these cases is presented in the accompanying table 1 and box graphs in figure 4. Each scenario was evaluated against an optimal solution determined by a mathematical solver and against a heuristic.The primary objective is naturally to match the optimal solution, but surpassing the heuristic also represents a significant achievement. Let's deep dive further into results and analyze the test environment.



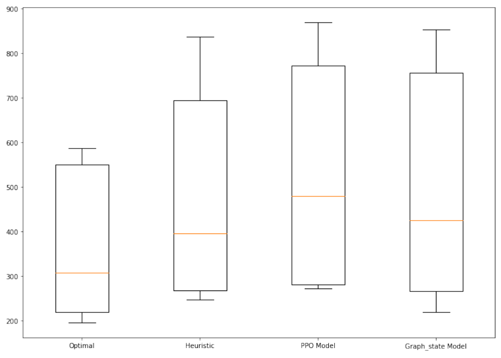
((a)) No flexibility - No setup times



((b)) No flexibility - with setup times



((c)) Flexibility - No setup times



((d)) Flexibility - with setup times

Figure 4: Here we can see 4 different distinct cases mentioned in the text. For each box graph, we have 4 base models as Optimal, Heuristic, PPO-Model, and GNN-Model.

| Configuration | Optimal | Heuristic | PPO-Model | GNN-Model | Fail Rate Optimum % | Fail Rate Heuristic % |
|---|---|---|---|---|---|---|
| No-Flex, No-setup, 1 stage | 654.25 | 654.25 | 654.25 | 654.25 | 0.00 | 0.00 |
| No-Flex, No-setup, 3 stages | 1241.5 | 1385 | 1417.25 | 1374 | 10.67 | -0.79 |
| No-Flex, Yes-setup, 1 stage | 509.5 | 510.75 | 622.25 | 611 | 19.92 | 19.63 |
| No-Flex, Yes-setup, 3 stages | 787.5 | 810.75 | 926.5 | 915.25 | 16.22 | 12.89 |
| Yes-Flex, No-setup, 1 stage | 318.75 | 352.5 | 394.25 | 381.5 | 19.69 | 8.23 |
| Yes-Flex, No-setup, 3 stages | 415 | 463 | 509.75 | 499.5 | 20.36 | 7.88 |
| Yes-Flex, Yes-setup, 1 stage | 226.5 | 272.75 | 293.75 | 266.25 | 17.55 | -2.38 |
| Yes-Flex, Yes-setup, 3 stages | 506 | 687.5 | 772.25 | 734 | 45.06 | 6.76 |

Table 1: Comparison of Optimal, Heuristic, PPO-Model, and GNN-Model results. For each configuration, four distinct random instances are generated, trained, tested, and evaluated against both the optimal and heuristic solutions. The numerical results presented here are the average outcomes from these four cases.

The initial observation that stands out is the scenario with no flexibility and no setup times, involving just a single stage. In this configuration, there is zero percent deviation from the reference solutions. This outcome is attributed to the simplicity of the setup, where having only one stage means that regardless of how orders are allocated, the completion time remains constant. Therefore, it is logical to exclude this case from further evaluation.

Upon reviewing the solutions overall, we notice a gap of 10 to 20 percent between the optimal solution and our GNN-Model. Although this is an improvement over random task assignments, the performance falls short of our expectations. The results suggest that our model needs additional refinements. Identified areas for improvement include the need for more thorough fine-tuning and enhancing the graph representation of the environment.

Our developed model incorporates three distinct neural networks, two of which form the actor-critic model, and the third serves as a graph neural network for state representation. These networks collectively have a large number of parameters that require tuning, including the learning rates for each network, specific actor-critic parameters such as gamma, the number of K epochs, the epsilon clip threshold, and the GAE lambda value. Additionally, structural parameters of the networks, such as the number of layers in each network, the internal hidden dimensions, and even the output dimension of the GNN, must also be optimized. While there are some guiding principles for setting these parameters, configuring them simultaneously presents a highly complex challenge.

One aspect of our project is the graph representation of states, where we have introduced an original approach that emphasizes resource structures over job information. Upon further examination of the features representing each node in our graph (which correspond to resources), we realized that some crucial order-related information was not as clearly encoded in the model as intended. While there are some references to this information, enhancing the feature construction to more explicitly incorporate job flexibility could improve our model's handling of such scenarios. This inadequacy likely contributes to the larger deviations from the optimal solution observed in configurations that involve flexibility. Interestingly, the heuristic model also shows limitations in managing flexibility, yet in some instances, our model performs comparably or even surpasses the heuristic results.

## 5.2 Model's ability to generalize

Another criterion for evaluation is the model's ability to generalize. After training the model with certain cases, we test it on entirely new ones to gauge its effectiveness on unseen scenarios. This metric is essential as it demonstrates the model's applicability to new instances, which often occurs in practical settings. The violin graph 5 included illustrates the performance of our model compared to the heuristic.

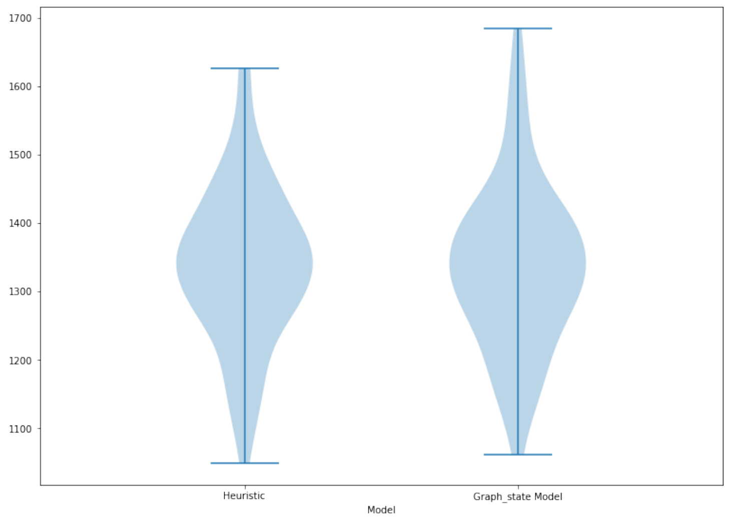


Figure 5: Violin graph to compare heuristic results with the result of our model.

As previously mentioned, we concentrate on one configuration where our model demonstrates potential. Specifically, we select the scenario with No Flexibility and No Setup times across three stages. This case was chosen because our model not only surpassed the heuristic performance but also showed a minimal gap to the optimal scenario. Another promising case is the configuration with Flexibility and Setup times across either 1 or 3 stages. This scenario represents the most complex configuration, where our model equaled the heuristic performance, although the gap to the optimal solution remains substantial.

In order to test our model's ability to generalize, we generated a significant number of random instances (120 in total). We train our model on 20 of these cases and assess its performance on the remaining 100. For this evaluation, as noted earlier, we exclusively use heuristic benchmarks to gauge the effectiveness of our model, as coming up with optimal solutions for all of the cases are costly and heuristic method sets a valid base for the comparison.

The violin plot serves as a visual representation to evaluate the generalization capabilities of the Graph State Model. The plot delineates the performance distribution between the heuristic method and the Graph State Model, shedding light on the model's effectiveness in handling unseen data and its potential for broader applicability.

The analysis of the violin plot is augmented by additional performance metrics that were gathered independently from the plot. Specifically, the Graph State Model demonstrates commendable generalization abilities, as it outperformed the heuristic in 55 out of the 100 cases tested. Additionally, the average performance gap between the heuristic and the model is recorded at -0.1731%. , suggesting that, on average, the Graph State Model not only meets but slightly surpasses the heuristic in terms of efficiency. This percentage gap, calculated by aggregating all the individual percentage gaps and averaging them over the cases, indicates a modest yet consistent edge of the Graph State Model over the heuristic.

Violin plots are particularly useful in such analyses as they provide a comprehensive view of the data distribution, highlighting the density and the range of outcomes. The broader base and tapered ends of the Graph State Models plot suggest a wider range of performance outcomes, with the potential for both high and low extremes. This variability points to the model's sensitivity to specific configurations or operational conditions, underscoring the importance of further refinement to enhance its predictability and stability across various scenarios.

In conclusion, the Graph State Model shows promising generalization capabilities, performing better than the heuristic in a majority of the cases. However, the spread of outcomes highlighted in the violin plot indicates room for improvement. Enhancing the model's consistency across different operational scenarios could lead to more reliable and universally superior results. This analysis not only confirms the model's potential but also guides future adjustments and optimizations necessary for broader application.

# 6 Discussions and Conclusions

In this project, we developed and successfully implemented a simulation environment tailored to address the complexities of the Job Shop Scheduling Problem (JSSP). Leveraging the capabilities of Deep Reinforcement Learning (DRL), we incorporated the Proximal Policy Optimization (PPO) method to develop a foundational model aimed at optimizing job scheduling. To enhance the model's representation of dynamic production environments, a Graph Neural Network (GNN) layer was introduced, capturing the state as a graph. This adaptation facilitated greater flexibility in accommodating changes in resource setups, proving the utility of GNNs in representing complex operational relationships.

To evaluate our model, various setups were devised, each assessing different aspects of the JSSP under increasing complexity, specifically focusing on flexibility in resource selection and setup times. Our evaluation involved two distinct assessment processes: analyzing the gaps between the optimal solutions and our model's outcomes, and assessing the models ability to generalize across different instances. These evaluations highlighted that while our model can occasionally surpass heuristic benchmarks and exhibit near-optimal performance, particularly in simpler configurations without flexibility or setup times, it often falls short in more complex scenarios by 10 to 20 percent. This discrepancy indicates a need for further refinement in the models training and structural parameters.

The model's underlying complexity, derived from multiple neural networks with extensive parameter configurations, presents a significant challenge in achieving optimal efficiency. Key areas identified for improvement include enhancing the model's tuning process and refining its graph representation capabilities to better manage the intricacies of job scheduling scenarios.

Additionally, the model's current approach to encoding state information has shown limitations, particularly in scenarios requiring high flexibility. Addressing these deficiencies by more accurately representing crucial scheduling details in the model's architecture could lead to improved performance and closer alignment with optimal solutions. These enhancements will not only refine the models accuracy but also broaden its applicability and effectiveness across varying job scheduling environments. This ongoing refinement process is essential to advancing the model's capabilities and achieving a standard of performance that meets or exceeds established heuristic benchmarks.

Looking ahead, an intriguing avenue for future work involves further exploring the model's generalization capabilities in configurations characterized by both Flexibility and Setup times, particularly across configurations of 1 or 3 stages. This scenario, noted as the most complex, provides a unique testbed where our model has demonstrated the ability to match heuristic performance, despite a considerable gap to the optimal solutions. Future efforts could focus on adapting and refining the model specifically for these configurations, potentially through enhanced training strategies or more sophisticated neural network architectures. This targeted approach aims not only to narrow the gap with optimal solutions but also to solidify the models effectiveness in complex, dynamic scheduling environments, ultimately improving decision-making precision and adaptability.

To sum up, there is substantial scope for refining the system. Further tuning of hyperparameters, experimenting with different network architectures, and integrating more sophisticated learning strategies could potentially enhance performance. Additionally, incorporating more detailed feedback mechanisms to adjust learning processes in real-time could address some of the shortcomings encountered in this initial implementation. Continued research and experimentation are essential to unlock the full potential of combining DRL with GNN for complex scheduling tasks, aiming to eventually surpass traditional methods in both efficiency and adaptability.

## References

[1] Junyoung Park, Jaehyeong Chun, Sang Hun Kim, Youngkook Kim, and Jinkyoo Park. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. *International Journal of Production Research*, 59(11):33603377, January 2021.

[2] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning, 2020.

[3] Wen Song, Xinyang Chen, Qiqiang Li, and Zhiguang Cao. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 19(2):1600–1610, 2023.

[4] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[5] Rocío Mercado, Tobias Rastemo, Edvard Lindelöf, Günter Klambauer, Ola Engkvist, Hongming Chen, and Esben Jannik Bjerrum. Graph networks for molecular design. *Machine Learning: Science and Technology*, 2(2):025023, mar 2021.

[6] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.