Deep Q-Networks for Hyperparameter Tuning in Flow Shop Scheduling Problem

Marouane Benbetka[1], Ahmed Rayane Kebir[1], Merwan Bekkar[1], Ramzi Baaguigui[1], Mohamed Chakib Bourzag[1], Ines Manel Boumazouza[1], Ibtissam Fatma Chouchou[1], Malika Bessedik[12]

[1] *Ecole Nationale Superieure d'Informatique (ESI), BP 68M - 16270 Oued Smar, Alger, Algeria*

[2] *Laboratoire des Methodes de Conception de Systemes (LMCS), BP 68M - 16270 Oued Smar, Alger, Algeria*

Abstract

The Flow Shop scheduling problem, a classic optimization challenge, often requires effective metaheuristic approaches to find optimal or near-optimal solutions within a reasonable computational time. Common metaheuristics such as Genetic Algorithms (GA) and Simulated Annealing (SA) depend heavily on their hyperparameter settings, which significantly influence their performance. However, the selection of these hyperparameters is not straightforward and typically requires extensive manual tuning. This paper introduces a novel hyper-heuristic framework that employs Deep Q-Networks (DQNs) to automate the tuning process of these hyperparameters. Our approach involves training individual DQN models for each metaheuristic to determine optimal hyperparameter settings based on the state of the problem instance. We construct our DQN agents with unique models and optimizers for each metaheuristic, enabling them to learn from specific interactions and adapt their strategies accordingly. We demonstrate the efficacy of our method through extensive experiments where DQNs trained on smaller instances are generalized to larger, more complex instances. This study bridges the gap between reinforcement learning and combinatorial optimization, providing a scalable and efficient solution for hyperparameter tuning in metaheuristic algorithms.

Key words. Flow Shop scheduling, metaheuristics, hyperparameter tuning, Deep Q-Network, Genetic Algorithms, Tabu Search, Simulated Annealing, reinforcement learning, hyper-heuristic.

1. INTRODUCTION

The Flow Shop scheduling problem is a fundamental combinatorial optimization challenge [3] that has garnered significant attention in the operations research and industrial engineering fields due to its practical relevance in manufacturing and service industries. The problem involves sequencing a set of jobs through a series of machines in the same order, aiming to minimize the total processing time, the completion time of the last job, or other operational objectives. Due to its NP-hard nature, finding optimal solutions for moderately sized instances is computationally infeasible with exact methods, especially as the size and complexity of the problem increase[17].

Traditional approaches to the Flow Shop problem started with heuristic methods, which are designed to be problem-specific and provide good solutions in a reasonable amount of time without guaranteeing optimality [16]. Examples include the Johnson's rule for two-machine scenarios [2], which provides an optimal solution under specific conditions. As problem complexity grows, these heuristics are often outperformed by more advanced techniques.

Metaheuristic algorithms, which are higher-level frameworks not limited to a specific problem [26], have been developed to find high-quality solutions for complex optimization problems. These can be broadly categorized into two groups: population-based [4] and single-solution-based metaheuristics [14]. Population-based methods, such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) [28], operate on a set of potential solutions, allowing for a diverse search of the solution space and reducing the risk of converging to local optima. Single-solution-based methods, such as Simulated Annealing (SA) [22] and Tabu Search (TS) [5], iteratively improve a single solution by exploring its

neighborhood, which can be more efficient in terms of memory and computational resources but may require more careful tuning to avoid local minima.

Despite their effectiveness, a significant limitation of metaheuristics is the dependency on hyperparameters [10] , which control the behavior and performance of the algorithm. The selection of appropriate hyperparameter values is crucial as it can drastically affect the convergence speed and the quality of the final solution. Traditionally, this tuning process has been performed manually through trial and error, grid search, or random search, which are time-consuming and often inefficient.

Recent advances in artificial intelligence have introduced the potential of leveraging machine learning techniques to optimize these hyperparameters [15]. In particular, reinforcement learning (RL) has shown promise in automating decision-making processes. Deep Q-Networks (DQNs) [12], an RL method combining Q-learning [29] with deep neural networks [30], provide a powerful tool for learning optimal policies over high-dimensional input spaces, making them suitable for the hyperparameter tuning task in complex metaheuristics. This paper proposes a novel hyper-heuristic framework utilizing DQNs to autonomously determine the optimal settings for various metaheuristic algorithms when solving the Flow Shop problem.

In the following sections, we will review the literature on the use of metaheuristics for the Flow Shop problem, describe our DQN-based hyper-heuristic model in detail, and present experimental results that demonstrate the effectiveness of our approach in optimizing the performance of metaheuristics across different problem instances.

2. PROBLEM STATEMENT

The Flow Shop Scheduling Problem (FSP) involves scheduling a set of n jobs on m machines. Each job must be processed in the same sequence on all machines, meaning that each job j must first be processed by machine 1, then by machine 2, and so on up to machine m. The typical goal is to minimize the makespan, which is the total time required to complete all jobs, denoted by $C_{\max}$ [11].

The mathematical formulation of the FSP can be presented as follows:

Let p_{ij} be the processing time of job j on machine i. The sequence of jobs is denoted by π, where $\pi(j)$ is the position of job j in this sequence. The start times S_{ij} and completion times C_{ij} for each job j on each machine i must satisfy the following constraints:

Initial Completion Time:

$$C_{i,\pi(1)} = \sum_{k=1}^{i} p_{k,\pi(1)}, \qquad \text{for } i = 1, \ldots, m,$$

Job Scheduling Constraints:

Start Time Constraint: Ensures that each job begins only after the previous job has completed on the same machine and after it itself has finished on the preceding machine.

$$S_{i,\pi(j)} = \max(C_{i-1,\pi(j)}, C_{i,\pi(j-1)}),$$
$$\text{for } j = 2, \ldots, n, \text{ and } i = 1, \ldots, m.$$

Completion Time Constraint: Determines the completion time of each job based on its start time and processing time on the current machine.

$$C_{i,\pi(j)} = S_{i,\pi(j)} + p_{i,\pi(j)},$$
$$\text{for } j = 1, \ldots, n, \text{ and } i = 1, \ldots, m.$$

These formulas ensure that each job starts on a machine only after it has been completed on the previous machine and that the jobs are processed in the order defined by π on all machines. The objective is to minimize $C_{\max} = C_{m,\pi(n)}$, which is the completion time of the last job on the last machine.

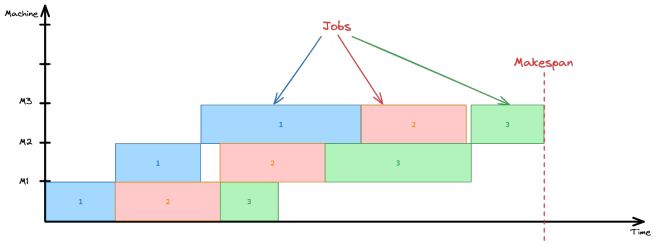

Fig. 1: FSP execution Plan

3. LITERATURE REVIEW

Optimization methods for the FSP can be categorized into two main types: exact methods and approximate methods, each with subcategories relevant to the FSP [31].

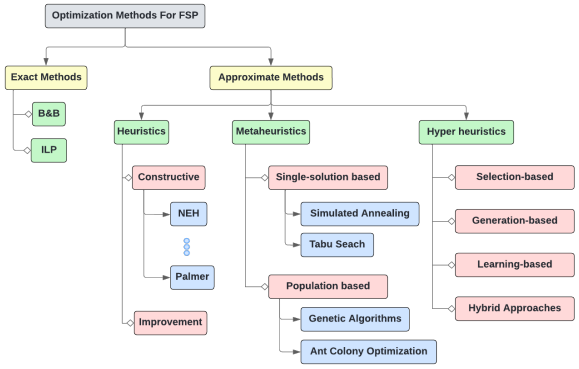

Fig. 2: Optimization methods for the Flow Shop Scheduling Problem

3.1. Exact Methods

Exact methods provide a guaranteed optimal solution for the FSP but are typically limited to small instances due to computational complexity. Among these methods are:

Branch and Bound: This method involves exploring all possible branches of solutions and pruning branches that do not meet the criteria early in the computation process [13], thereby reducing the search space.

Integer Linear Programming (ILP): This approach models the problem with linear inequalities and integral variables. It is highly effective when combined with cutting planes and branch-and-cut algorithms to solve smaller instances efficiently [6].

3.2. Approximate Methods

Due to the limitations of exact methods for large instances, approximate methods are more commonly used for the FSP. These methods are divided into two subcategories: heuristics and metaheuristics.

3.2.1. Heuristics

Heuristics are quick solution methods that do not guarantee optimality but are effective in obtaining good quality solutions rapidly. For the FSP, examples include:

- **Constructive heuristics** (e.g., Nawaz-Enscore-Ham (NEH) algorithm [19]): This method sequentially builds a job schedule by inserting the next job at the position that results in the minimum increase of the schedule length.
- **Improvement heuristics** (e.g., Local Search [21]): These heuristics start with an initial solution and iteratively make small changes to improve the solution, such as swapping job positions or reordering parts of the schedule.

3.2.2. Metaheuristics

Metaheuristics provide a flexible framework for exploring the search space and are particularly useful for large instances of the FSP. They include:

- **Single-solution based metaheuristics** (e.g., Simulated Annealing (SA) [27] and Tabu Search (TS) [5]): SA uses probabilistic techniques to escape local optima by accepting worse solutions early in the process, while TS uses a memory structure to avoid cycling back to previously visited solutions.
- **Population-based metaheuristics** (e.g., Genetic Algorithms (GAs) [18] and Ant Colony Optimization (ACO) [1]): GAs simulate natural evolutionary processes such as selection, crossover, and mutation, whereas ACO mimics the pheromone trail-laying and following behavior of ants to find optimal paths through graphs.

3.3. Hyper-Heuristics

The Flow Shop Scheduling Problem has attracted a wide array of hyper-heuristic approaches due to its complexity and practical significance in industrial settings. Hyper-heuristics, being a layer above heuristics and metaheuristics, aim to provide generalized problem-solving frameworks by selecting or generating heuristics based on the problem context. This section reviews some of the notable hyper-heuristic methods developed specifically for the FSP, illustrating their strategies and effectiveness.

Selection-based Hyper-heuristics: These methods focus on choosing the best heuristic from a predefined set based on the current state of the problem. For instance, a selection-based hyper-heuristic [7] that adaptively chooses among a set of simple dispatching rules and more complex metaheuristics

depending on the phase of the search. The study demonstrated improved consistency in finding near-optimal solutions across different FSP instances compared to using any single heuristic alone.

Generation-based Hyper-heuristics: In contrast to selection-based methods, generation-based hyper-heuristics create new heuristics from components of existing ones. An example using a genetic programming approach to evolve new heuristics specifically tailored for the FSP [9]. This method proved effective in adapting to the characteristics of different FSP instances, showcasing the potential of evolutionary techniques in heuristic generation.

Learning-based Hyper-heuristics: These approaches incorporate machine learning techniques to adapt and improve their heuristic selection or generation strategies over time [23].

Hybrid Approaches: Hybrid hyper-heuristics combine elements of selection, generation, and learning strategies to leverage their respective strengths [24].

These approaches are continually adapted and improved to meet the specific challenges of the FSP, often incorporating innovations from various fields of artificial intelligence, including machine learning for parameter tuning and strategy adaptation.

3.4. Limitations of Traditional Methods

While the traditional optimization methods outlined above have proven effective for various instances of the FSP, they each come with inherent limitations that can hinder their practical applicability, especially in complex or large-scale scenarios. Exact methods such as Branch and Bound and Integer Linear Programming, although precise, suffer from exponential growth in computational time and memory requirements as the problem size increases. This renders them impractical for modern industrial applications, where job sequences and machine numbers can be extensive.

Heuristic approaches, while generally faster and more scalable than exact methods, often do not guarantee proximity to the optimal solution. These methods typically provide quick solutions but can vary significantly in quality, with results often far from the best possible outcomes. Moreover, heuristics are usually problem-specific, meaning they are tailored for specific scenarios and may not perform well when applied to different types of problems or when the problem parameters change. This inconsistency and lack of generalizability make heuristics less reliable for applications where solution quality and adaptability across various scenarios are critical.

Furthermore, metaheuristics, which are designed to overcome some of the limitations of simpler heuristics, introduce their own complexities. They often involve a large number of hyperparameters that need to be finely tuned to achieve the best results. The performance of metaheuristic algorithms is heavily dependent on this tuning process, which can be both time-consuming and technically challenging. The need to manually adjust these parameters adds an additional layer of complexity, making these methods less accessible for non-experts and cumbersome in dynamic or evolving operational contexts.

Given these challenges, this paper introduces a novel hyper-heuristic [8] approach that leverages the capabilities of a Deep Q-Network (DQN) to autonomously determine the optimal hyperparameters for various metaheuristics. Our approach aims to bridge the gap between the robustness of metaheuristic strategies and the efficiency and adaptability required in practical applications. By automating the hyperparameter tuning process, our hyper-heuristic not only enhances the performance of underlying metaheuristics but also significantly reduces the expertise required to deploy these powerful optimization tools in real-world scenarios. This methodology represents a substantial advancement in the field of optimization, offering a scalable, efficient solution to complex scheduling problems such as the FSP.

4. PROPOSED SOLUTION

In response to the challenges posed by traditional optimization techniques for the FSP, especially their limited scalability and the labor-intensive nature of hyperparameter tuning, we propose a sophisticated solution utilizing DQNs. This approach leverages recent advances in reinforcement learning to automate and enhance the hyperparameter selection process for various metaheuristic algorithms.

Our proposed framework introduces a DQN architecture specifically designed to interact with the FSP environment. By applying a reinforcement learning model, the DQN agent systematically learns to evaluate and modify hyperparameters in real-time, thereby optimizing the performance of metaheuristic algo-

rithms under varying conditions. This automated learning process is both scalable and efficient, addressing key shortcomings of manual parameter tuning and heuristic stagnation.

The subsequent sections detail the configuration of our DQN model, including the state representation, action space, and reward structure tailored for the metaheuristic optimization landscape. We also discuss the training protocol and the empirical results obtained from extensive simulations, illustrating the capability of our DQN approach to consistently outperform traditional methods in terms of solution quality and computational resources.

4.1. Components of the Hyper-Heuristic Architecture

Our DQN-based hyper-heuristic architecture for the FSP leverages a sophisticated interaction between high-level strategic decision-making and low-level heuristic operations. This section outlines the architecture's components [33] as they specifically apply to optimizing metaheuristics for the FSP:

- **High-Level Strategy:**
    1. *Choice Function* - This component is embodied by the decision-making capability of the DQN agent, which uses a neural network model to evaluate and select the most effective hyperparameter configurations for the metaheuristics. The network predicts the expected reward of choosing specific configurations, effectively guiding the search process towards optimal solutions.
    2. *Internal Memory* - Central to our learning mechanism is the agent's replay memory, which stores experiences in the form of state-action-reward transitions. This memory allows the DQN to learn from past actions and their outcomes, facilitating the refinement of its policy over time through experience replay.
- **Low-Level Heuristics:**
    1. The actual metaheuristic algorithms (Genetic Algorithm, Simulated Annealing, Tabu Search) serve as the low-level heuristics. These are directly applied to the FSP instances, with their operational parameters dynamically adjusted by the high-level strategy based on the ongoing learning process.
- **Domain Barrier (Filter):**
    1. *Standard Interface* - The interface between the high-level strategy and the low-level heuristics is standardized through the DQN's action space and the environment's structure. The DQN agent operates abstractly, selecting actions without needing to understand the intricate details of how the heuristics manipulate the problem space. This separation ensures that the agent can be applied to different FSP instances or potentially other scheduling problems with minimal adjustments.

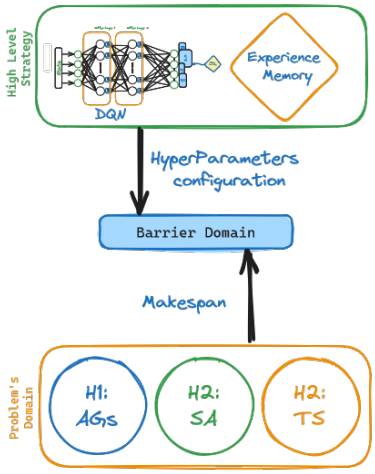

Fig. 3: Global Architecture of the proposed hyper-heuristic

This structured approach, dividing the architecture into high and low levels with a clear domain barrier, enables our DQN-based system to efficiently learn and optimize metaheuristic parameters. It enhances the adaptability and performance of the solution, making it robust against the varied challenges presented by different FSP instances. By continually updating the learning model through interactions with the environment, the system not only adapts to the specific nuances of the FSP but also improves its decision-making strategy over time.

4.2. Architecture Overview

The core of our solution is the DQN model, which comprises:

- **Deep Q-Network Model:** A neural network that estimates the potential reward (Q-value) for each action in a given state [12]. The architecture consists of:
  1. An input layer that represents the state of the FSP.
  2. Two hidden layers with 128 neurons each, using ReLU activation functions.
  3. An output layer providing Q-values for each possible action, representing different hyperparameter configurations for the metaheuristics.

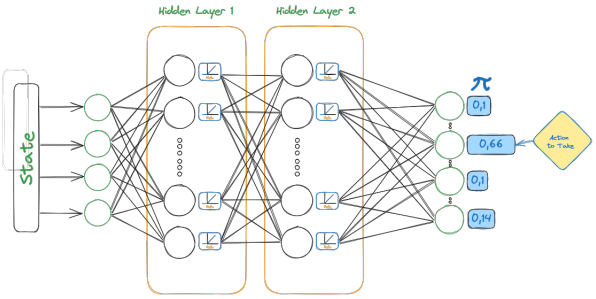

Fig. 4: DQN Model Architecture

- **Agent and Memory:** Utilizes the model to choose actions and stores experiences in a replay memory, essential for learning from past actions by reducing temporal correlation.
- **Learning Mechanism:** Employs the Adam optimizer [32] to minimize the loss, which is the mean squared error between the predicted Q-values and the target values, updated by the Bellman equation:

$$\text{Loss} = \mathbb{E}\left[(r + \gamma \max_{a'} Q(s', a') - Q(s, a))^2\right]$$

4.3. Action Decoding and Hyperparameter Configuration

Actions within the DQN model represent selections of hyperparameters for specific metaheuristics. Each action, when taken, updates the current settings of a metaheuristic to potentially optimize the scheduling outcome. The chosen hyperparameters for each metaheuristic are based on their influence on the algorithms' performance:

- **Genetic Algorithm (GA) [18]:**
  - *Population Size:* Affects diversity and quality of solutions. Large populations offer diversity but increase computational load.
  - *Mutation Rate:* Balances exploration of new areas in the solution space against retention of good solutions.
  - *Crossover Rate:* Crucial for combining good traits from different solutions to explore new, potentially superior, solutions.
  - *Selection Method:* Influences how individuals are chosen to reproduce, affecting the speed of convergence and diversity preservation.
- **Simulated Annealing (SA) [22]:**
  - *Cooling Rate:* Controls the rate at which the temperature decreases, affecting the exploration-exploitation balance.
  - *Initial and Final Temperature:* Set the starting conditions and the termination condition of the annealing process, impacting the thoroughness of the search.
- **Tabu Search (TS)[5]:**
  - *Tabu List Size:* Determines the memory of the search, preventing cycles and encouraging exploration of new areas.
  - *Max Iterations and Neighborhood Size:* Control the depth and breadth of the search space exploration.

Dynamic Hyperparameter Adjustment Method:

Hyperparameter adjustments within our DQN framework are made dynamically, with each action directly mapping to a predefined set of hyperparameter configurations. This method employs modular arithmetic and division operations to systematically derive variations in settings, ensuring a broad and effective exploration of the configuration space.

For instance, consider the hyperparameter settings for the Genetic Algorithm (GA):

- **Population Size:** Adjusted using the formula $50 + (action\_index\%5) \times 10$. If the *action_index* chosen by the DQN is 7, the population size would be calculated as $50 + (7\%5) \times 10 = 70$. This formula allows the system to explore different levels of population diversity, impacting the

genetic diversity within the population and the overall performance of the GA.

- **Mutation Rate:** Set by $0.05 + (action\_index\%4) \times 0.01$. For the same $action\_index$ of 7, the mutation rate becomes $0.05 + (7\%4) \times 0.01 = 0.08$. This rate controls the introduction of new genetic variations into the population, crucial for exploring new areas of the solution space.
- **Crossover Rate:** Configured as $0.6 + (action\_index//5) \times 0.05$, which for $action\_index$ 7 calculates to $0.6 + (7//5) \times 0.05 = 0.65$. This rate is vital for combining beneficial traits from parent solutions to produce potentially superior offspring.

Similar principles apply to the adjustment of parameters in Simulated Annealing and Tabu Search.

Theoretical Justification: The selection and adjustment of these hyperparameters are based on heuristic techniques that aim to balance exploration and exploitation within the search space. By dynamically adjusting these parameters based on learning from past experiences, our DQN architecture seeks to find an optimal balance that can adapt to the complexities and nuances of different FSP instances, potentially leading to consistently superior solutions. This method also allows the system to generalize its learning across various instances and configurations, improving its robustness and efficiency over time.

4.4. High-Level Functionality and Data Structures

The DQN agent operates within a custom-designed environment, managing the state space (current job-machine matrix) and action space (metaheuristic hyperparameter settings). The agent's actions directly influence the hyperparameter configurations applied to solve the FSP, with the environment providing feedback in the form of the new state and reward based on the makespan.

4.5. Learning and Adaptation Layer

The agent's continuous learning and adaptation are facilitated by:

- **Continuous Learning:** Interaction with the FSP environment refines hyperparameter settings over many episodes, using stored experiences to minimize loss and adjust model parameters.
- **Adaptation Mechanism:** Through backpropagation and learning adjustments, the agent adapts to dynamic changes, enhancing decision-making about hyperparameter adjustments to consistently minimize makespan.

4.6. Algorithmic Outline of the DQN Hyper-Heuristic Framework

To provide a clear and concise representation of our proposed solution, we present a simplified version of the DQN-based hyper-heuristic algorithm. This pseudocode encapsulates the core processes of our approach, highlighting the cyclic nature of the learning and optimization that occurs within the DQN framework. The algorithm focuses on the key steps of initializing the model, interacting with the environment, updating based on experience, and applying learned strategies to new instances of the FSP. It distills the essence of the more complex operations into a format that is accessible and straightforward, making it easier to grasp the operational flow of the hyper-heuristic system.

Algorithm 1 Simplified DQN-based Hyper-Heuristic for FSP Optimization

Input: Initial FSP instance
Output: Optimized metaheuristic configuration
Initialize DQN model with random weights
Define action space for hyperparameters
Initialize replay memory
for each training episode **do**
    Initialize environment with FSP instance
    while not done **do**
    Choose action using ϵ-greedy policy from DQN
    Execute action in environment (apply metaheuristic)
    Observe reward and new state
    Store experience in replay memory
    Update DQN model by sampling from memory
    end while
end for
For new FSP instance:
Evaluate using optimized metaheuristic parameters
Return solution and performance metrics

This DQN-based hyper-heuristic framework not only automates the hyperparameter tuning process but also significantly enhances the performance and robustness of metaheuristic algorithms in solving the FSP, demonstrating a considerable advancement over traditional approaches.

5. Experimental Results

This section presents a detailed analysis of various experiments conducted to optimize the performance of the DQN-based hyper-heuristic framework for solving the FSP. Each experiment focuses on a specific aspect of the DQN algorithm—number of episodes, gamma factor, epsilon greedy strategy, and learning rate—to determine the best settings for achieving optimal results. The tests were applied on a standard set of FSP instances to ensure consistency and reliability in the findings.

5.1. Evaluation Metric

Objective: To assess the performance of the DQN framework reliably and consistently across different experimental settings, we employ the Relative Deviation Percentage (RDP) as our primary evaluation metric.

Definition: The RDP is calculated as follows:

$$\text{RDP} = \left(\frac{\text{Obtained Makespan} - \text{Upper Bound}}{\text{Upper Bound}} \right)$$

where the *Obtained Makespan* is the makespan produced by the DQN for a given FSP instance, and the *Upper Bound* is the best known makespan achieved by any method for the same instance up to this point. This measure allows us to quantify the efficiency of the DQN in terms of its closeness to the best-known solutions, providing a standardized way to compare its performance across various configurations and instances.

Rationale: Using RDP enables us to measure the percentage by which the DQN's solutions deviate from the best available benchmarks. This metric is particularly useful in highlighting improvements over the state-of-the-art methods and in illustrating the effectiveness of various parameter adjustments within the DQN framework.

Execution Time: We also record and analyze the execution time for each experiment to evaluate the computational efficiency of our DQN framework. This aspect is particularly vital for understanding the practical viability of our approach in real-time and large-scale industrial environments.

5.2. Testing Environment

To ensure that the results of our experiments are reproducible and reflective of typical high-performance computational settings, the following hardware specifications were used for all simulations and testing:

Component	Specification
Processor Model	Intel Core(TM) i7-10750H CPU
Processor Frequency	2.60GHz, Turbo up to 5.0GHz
Installed RAM	16.0 GB
Graphics Card	NVIDIA GeForce GTX 1650 Ti

Table 1: Hardware Specifications of the Testing Environment

Following this standardized evaluation approach, we proceed to detailed experimental analyses focusing on key aspects of the DQN's configuration.

5.3. Benchmark Dataset

Context: The Taillard benchmark [25] problems are a standard set of instances commonly used in the evaluation of algorithms for the Flow Shop Scheduling Problem. These benchmarks are chosen because they represent a variety of problem sizes and complexities, providing a comprehensive challenge for our optimization techniques.

Details: Below is a table of selected Taillard benchmark instances that will be used in our experiments. This table lists each instance by name, number of jobs, number of machines, and the best known makespan up to this point, which serves as the upper bound in our RDP calculations.

Instance	Jobs	Machines	Upper Bound
Tai01	20	5	1278
Tai02	50	5	2724
Tai03	100	5	5493
Tai04	200	10	10868
Tai05	500	20	26189

Table 2: Taillard Benchmark Instances Used in Experiments

Using these benchmark instances ensures that our findings are comparable with other studies and provides a solid foundation for assessing the effectiveness and efficiency of our proposed method across different scales and difficulties.

5.4. Number of Episodes Analysis

Objective: To determine the optimal number of training episodes required for the DQN model to converge to a stable solution.

Methodology: The DQN was trained on small FSP instances with varying numbers of episodes, ranging from 100 to 1000, in increments of 100.

Results:

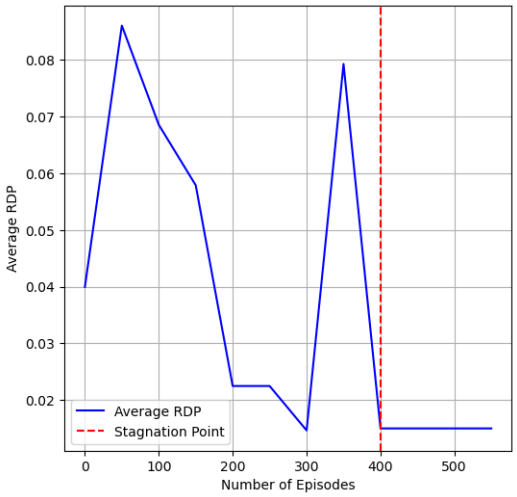

Fig. 5: Impact of varying number of episodes on DQN performance

Discussion: The results indicate that increasing the number of episodes generally improves the solution quality up to a certain point, beyond which the gains plateau, suggesting diminishing returns on additional training beyond 400 episodes.

5.5. Gamma Factor Analysis

Objective: To find the best gamma (discount factor) that balances the immediate and future rewards effectively.

Methodology: Tested gamma values ranged from 0.75 to 0.99 in increments of 0.01.

Results:

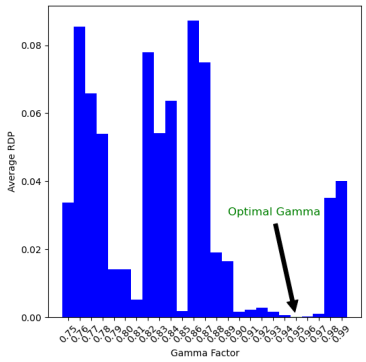

Fig. 6: Effect of gamma factor on the performance of DQN

Discussion: The optimal gamma value was found to be 0.95, providing the best compromise between short-term gains and long-term planning, enhancing overall solution stability and quality.

5.6. Epsilon Greedy Strategy Analysis

Objective: To optimize the exploration-exploitation balance in the learning process using the epsilon greedy strategy.

Methodology: The epsilon values were varied from 0.1 to 1.0 in increments of 0.1 to assess their impact on exploration and exploitation.

Results:

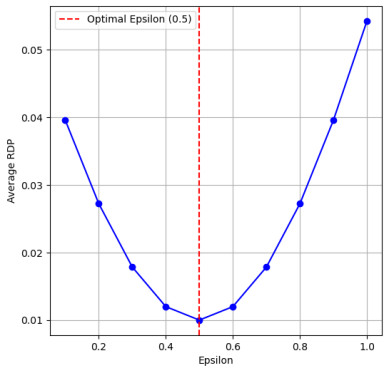

Fig. 7: Influence of epsilon values on exploration and exploitation balance

Discussion: An epsilon value of 0.5 was found to yield the best results, allowing sufficient explo-

ration of the action space without compromising the exploitation of known good actions.

5.7. Learning Rate Analysis

Objective: To identify the most effective learning rate for the DQN model's convergence and learning efficiency.

Methodology: Learning rates were tested from 0.001 to 0.1 in logarithmic steps.

Results:

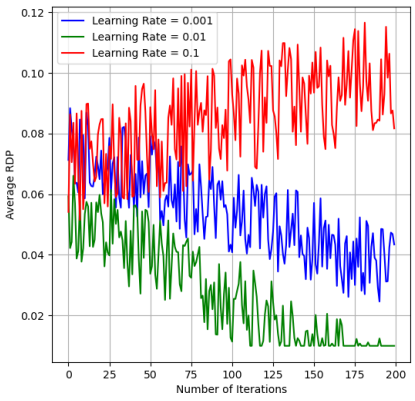

Fig. 8: Effects of different learning rates on DQN training dynamics

Discussion: A learning rate of 0.01 proved optimal, facilitating rapid convergence without causing the instabilities associated with higher rates.

These experiments collectively enhance our understanding of how different parameters affect the training and performance of DQN models in complex scheduling environments. The insights gained not only improve the efficacy of our approach but also contribute to the broader field of machine learning in combinatorial optimization settings.

5.8. Comparative Analysis

Objective: To evaluate the effectiveness of our DQN-based hyper-heuristic by comparing it against standard heuristic and metaheuristic approaches on selected Taillard benchmark instances (tai01, tai02, tai03).

Methodology: This analysis involves comparing our DQN-optimized solutions against those obtained using traditional approaches, including NEH, Palmer's heuristic, and both default and DQN-optimized parameters for Tabu Search and Simulated Annealing. Each method is tested on the same instances to ensure fairness and consistency in evaluation.

Competing Methods:

- **NEH** [19]: A well-known heuristic for FSP.
- **Palmer's Heuristic** [20]:Palmer's Heuristic another popular heuristic based on slope index calculations.
- **TS** [5]:Tabu Search with Default Parameters, a robust local search technique using a tabu list to avoid cycles.
- **TS-DQN**: Tabu Search enhanced by our DQN-based hyper-parameter tuning.
- **SA** [22]: Simulated Annealing with Default Parameters probabilistic technique for approximating the global optimum of a given function.
- **SA-DQN**: Simulated Annealing using parameters optimized via DQN.
- **GA** [18]:Genetic Algorithm with Default Parameters a robust evolutionary algorithm widely used for complex optimization problems.
- **GA-DQN**: The Genetic Algorithm enhanced by hyperparameters tuned via our DQN approach.

Results and Figures: For each of the benchmark instances, the performance of each method in terms of makespan is visually compared using bar graphs. The figures show the makespan achieved by each method, highlighting the efficiency and effectiveness of the DQN-optimized approaches.

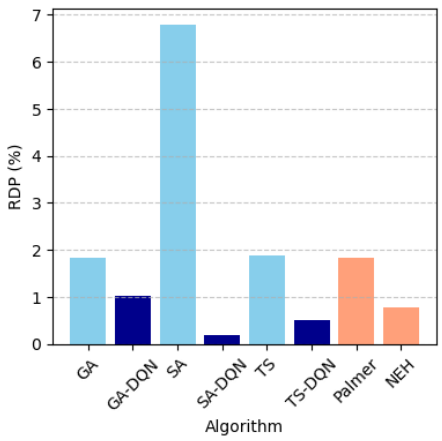

Fig. 9: Comparison of solution quality across different methods for instance tai02

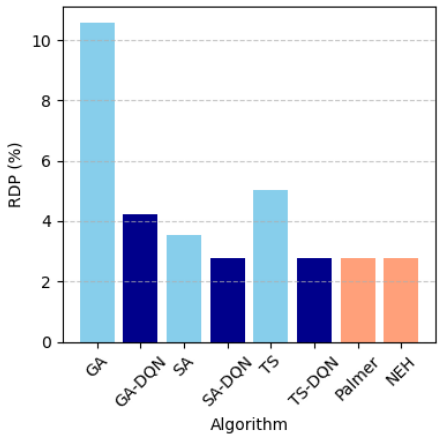

Fig. 10: Comparison of solution quality across different methods for instance tai03

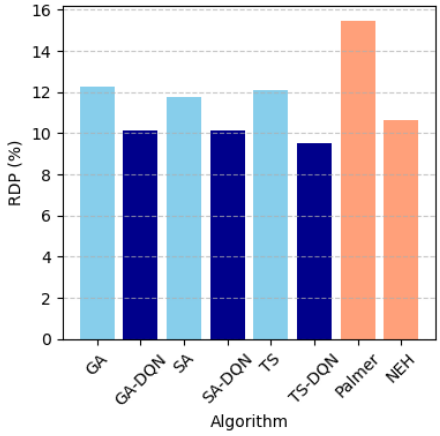

Fig. 11: Comparison of solution quality across different methods for instance tai05

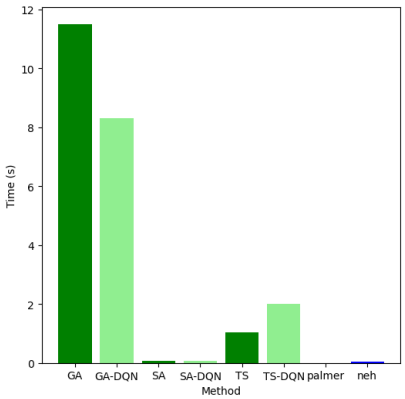

Fig. 12: Comparison of Methods and Their Execution Times for tai04

Discussion:

This comparative analysis underscores the strengths and limitations of each method, particularly focusing on the improvements in solution quality, as measured by the Relative Deviation Percentage (RDP). The introduction of a Deep Q-Network (DQN) to optimize heuristics and metaheuristics demonstrates significant potential in refining solution quality across various instances of the Flow Shop Scheduling Problem (FSP).

The RDP metric, which evaluates the efficiency of the DQN in achieving solutions close to the best-known outcomes, reveals that DQN optimization substantially enhances the solution quality. This is particularly evident in cases where traditional approaches like NEH and Palmer's heuristic, while fast, typically result in higher makespans. The DQN-enhanced methods, in contrast, consistently achieve lower RDP values, indicating closer proximity to optimal solutions and thereby highlighting the effectiveness of the DQN framework in improving the precision of the scheduling solutions.

While execution time is an important consideration, the primary focus of this analysis is on the quality improvements brought about by DQN. For example, Genetic Algorithm and Tabu Search, when optimized with DQN, not only maintain reasonable execution times but also show marked improvements in RDP scores. This suggests a significant enhancement in the ability to find superior solutions without excessively compromising computational efficiency.

These findings highlight the dual advantage of integrating DQN into traditional methods: significantly improved solution quality, as quantified by RDP, along with maintained or acceptable changes in execution times. This balance is crucial for practical applications where optimal performance and operational efficiency are both key.

Overall, the use of DQN not only propels heuristics and metaheuristics towards achieving outputs that are much closer to the theoretical bests but also illustrates a methodological advancement in tackling complex optimization challenges like the FSP, setting a benchmark for future explorations in this domain.

6. CONCLUSION

This paper presented a DQN-based hyper-heuristic framework for optimizing metaheuristics in the FSP. Our approach leverages advanced machine learning to automate hyperparameter tuning, significantly enhancing the efficiency and quality of solutions. A key advantage of our method is its ability to train on small instances and effectively generalize to larger, more complex problems, demonstrating robust scalability and practical applicability.

The DQN-enhanced methods consistently outperformed traditional approaches, achieving closer approximations to optimal solutions while maintaining reasonable computational times. These improvements highlight the potential of integrating deep learning techniques into operational research to address complex industrial challenges.

Looking forward, this research paves the way for applying similar strategies to other scheduling and optimization problems, expanding the potential of machine learning in operational settings. The success of this framework underscores its transformative potential, marking a significant step forward in the application of artificial intelligence in real-world optimization tasks. With the ongoing advancements in AI and machine learning, the possibilities for further enhancing and adapting these techniques to a broader range of applications appear both promising and limitless.

REFERENCES

[1] Kemal Alaykỳran, Orhan Engin, and Alper Döyen. Using ant colony optimization to solve hybrid flow shop scheduling problems. *The international journal of advanced manufacturing technology*, 35:541–550, 2007.

[2] Hamid Allaoui and AbdelHakim Artiba. Johnson's algorithm: A key to solve optimally or approximately flow shop scheduling problems with unavailability periods. *International Journal of Production Economics*, 121(1):81–87, 2009.

[3] Amr Arisha, Paul Young, and Mohie El Baradie. Flow shop scheduling problem: A computational study. 2002.

[4] Zahra Beheshti and Siti Mariyam Hj Shamsuddin. A review of population-based meta-heuristic algorithms. *Int. j. adv. soft comput. appl*, 5(1):1–35, 2013.

[5] M. Ben-Daya and M. Al-Fawzan. A tabu search approach for the flow shop scheduling problem. *European Journal of Operational Research*, 109(1):88–95, 1998.

[6] Kerem Bülbül and Philip Kaminsky. A linear programming-based method for job shop scheduling. *Journal of Scheduling*, 16:161–183, 2013.

[7] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. *Handbook of metaheuristics*, pages 449–468, 2010.

[8] Edmund K Burke, Matthew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches: revisited. *Handbook of metaheuristics*, pages 453–477, 2019.

[9] Marco Chiarandini, Mauro Birattari, Krzysztof Socha, and Olivia Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9:403–432, 2006.

[10] Bastien Chopard, Marco Tomassini, Bastien Chopard, and Marco Tomassini. Performance and limitations of metaheuristics. *An introduction to metaheuristics for optimization*, pages 191–203, 2018.

[11] Hamilton Emmons and George Vairaktarakis. *Flow shop scheduling: theoretical results, algorithms, and applications*, volume 182. Springer Science & Business Media, 2012.

[12] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. In *Learning for dynamics and control*, pages 486–489. PMLR, 2020.

[13] Jan Gmys, Mohand Mezmaz, Nouredine Melab, and Daniel Tuyttens. A computation-

ally efficient branch-and-bound algorithm for the permutation flow-shop scheduling problem. *European Journal of Operational Research*, 284(3):814–833, 2020.

[14] Essam H Houssein, Mohamed A Mahdy, Doaa Shebl, and Waleed M Mohamed. A survey of metaheuristic algorithms for solving optimization problems. In *Metaheuristics in machine learning: theory and applications*, pages 515–543. Springer, 2021.

[15] Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. Beyond manual tuning of hyperparameters. *KI-Künstliche Intelligenz*, 29:329–337, 2015.

[16] Douglas B Lenat. The nature of heuristics. *Artificial intelligence*, 19(2):189–249, 1982.

[17] Wenjun Li, Yang Ding, Yongjie Yang, R Simon Sherratt, Jong Hyuk Park, and Jin Wang. Parameterized algorithms of fundamental np-hard problems: A survey. *Human-centric Computing and Information Sciences*, 10:1–24, 2020.

[18] Ryohei Nakano and Takeshi Yamada. Conventional genetic algorithm for job shop problems. In *ICGA*, volume 91, pages 474–479, 1991.

[19] Muhammad Nawaz, E Emory Enscore, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[20] D. S. Palmer. Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. *Journal of the Operational Research Society*, 16(1):101–107, 03 1965.

[21] Quan-Ke Pan and Rubén Ruiz. Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, 222(1):31–43, 2012.

[22] Rob A Rutenbar. Simulated annealing algorithms: An overview. *IEEE Circuits and Devices magazine*, 5(1):19–26, 1989.

[23] Zhongshi Shao, Weishi Shao, and Dechang Pi. Ls-hh: A learning-based selection hyper-heuristic for distributed heterogeneous hybrid blocking flow-shop scheduling. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(1):111–127, 2022.

[24] Jorge A Soria-Alcaraz, Gabriela Ochoa, Marco A Sotelo-Figeroa, and Edmund K Burke. A methodology for determining an effective subset of heuristics in selection hyper-heuristics. *European Journal of Operational Research*, 260(3):972–983, 2017.

[25] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[26] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[27] Peter JM Van Laarhoven, Emile HL Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations research*, 40(1):113–125, 1992.

[28] Dongshu Wang, Dapei Tan, and Lei Liu. Particle swarm optimization algorithm: an overview. *Soft computing*, 22:387–408, 2018.

[29] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[30] Yu-chen Wu and Jun-wen Feng. Development and application of artificial neural network. *Wireless Personal Communications*, 102:1645–1656, 2018.

[31] Hegen Xiong, Shuangyuan Shi, Danni Ren, and Jinjin Hu. A survey of job shop scheduling problem: The types and models. *Computers & Operations Research*, 142:105731, 2022.

[32] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*, pages 1–2. Ieee, 2018.

[33] Fuqing Zhao, Shilu Di, Jie Cao, Jianxin Tang, et al. A novel cooperative multi-stage hyper-heuristic for combination optimization problems. *Complex System Modeling and Simulation*, 1(2):91–108, 2021.