Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

NS4S: Neighborhood Search for Scheduling Problems via Large Language Models

Junjie Zhang, **Canhui Luo**, **Zhouxing Su**, **Qingyun Zhang**, **Zhipeng Lü**, **Junwen Ding*** and **Yan Jin***
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
{junjie_zhang, luocanhui, suzhouxing, qingyun_zhang, zhipeng.lv, junwending}@hust.edu.cn, jinyan@mail.hust.edu.cn

Abstract

Large Language Models (LLMs) have emerged as a promising technology for solving combinatorial optimization problems. However, their direct application to scheduling problems remains limited due to the inherent complexity of these problems. This paper proposes an LLMs-based neighborhood search method that leverages LLMs to tackle the job shop scheduling problem (JSP) and its variants. The main contributions of this work are threefold. First, we introduce a novel LLMs-guided neighborhood evaluation strategy that guides local search by dynamically adjusting operation weights. Second, we develop a verification evolution (VeEvo) framework to mitigate the hallucination effects of LLMs, enabling the generation of high-quality heuristics for weight updates. Third, we integrate this framework with the weighted neighborhood evaluation strategy to effectively guide the search towards promising regions. Extensive experiments are conducted on 349 benchmark instances across three classical scheduling problems. The results demonstrate that our algorithm significantly outperforms existing state-of-the-art methods. For JSP, our algorithm reduces the average optimality gap from 10.46% to 1.35% on Taillard's instances compared to reinforced adaptive staircase curriculum learning. For flexible JSP (FJSP), it reduces the gap from 13.24% to 0.05% on Brandimarte's instances compared to deep reinforcement learning methods. Furthermore, for FJSP with sequence dependent setup time, our algorithm updates 9 upper bounds for benchmark instances.

1 Introduction

Job shop scheduling problem (JSP) is one of the most fundamental and extensively studied scheduling problems in combinatorial optimization, with significant applications in modern intelligent manufacturing systems and transportation logistics. The problem involves scheduling a set of jobs $J = \{J_1, J_2, ..., J_n\}$, where each job J_i consists of N_i operations $(O_{i1}, O_{i2}, ..., O_{iN_i})$ that must be processed sequentially on a set of machines $M = \{M_1, M_2, ..., M_m\}$ [Iklassov *et al.*, 2023]. Each operation O_{ij} must be processed on a specified machine with processing time PT_{ij}. In the flexible job shop scheduling problem (FJSP), each operation can be processed on any machine from its candidate machine set $M_{ij} \subseteq M$ with machine-dependent processing time PT_{ijk} [Yuan *et al.*, 2023]. For FJSP with sequence dependent setup time (FJSP-SDST), each operation requires setup time ST_{ijk} before processing, which depends on its machine predecessor [Oddi *et al.*, 2011]. All processing times and setup times are predetermined, and the objective is to minimize the maximum completion time (makespan) of all jobs.

The optimization of JSPs focuses on determining optimal operation sequences and processing times across multiple machines while considering various operational constraints. These scheduling problems are crucial in production planning and control, as they encompass complex aspects of resource allocation, task sequencing, and process optimization. Effective scheduling algorithms can significantly impact manufacturing performance by improving production efficiency, reducing operational costs, and minimizing production cycles. While the classical JSP focuses on fixed machine assignments, its variants such as FJSP and FJSP-SDST introduce additional complexity through flexible machine routing and sequence-dependent setup considerations, making them particularly relevant for modern manufacturing environments.

These scheduling problems belong to the class of NP-hard optimization problems, where computational complexity grows exponentially with problem size [Tian *et al.*, 2024]. While exact methods can guarantee optimal solutions, they become computationally intractable for large-scale instances, rendering them impractical for industrial applications. Although rule-based heuristics can rapidly generate feasible solutions, they often fail to provide high-quality solutions [Guo *et al.*, 2024]. Metaheuristics have thus emerged as the predominant research direction in scheduling optimization, offering an effective balance between solution quality and computational efficiency [Yao *et al.*, 2024].

Large Language Models (LLMs) offer several compelling advantages for addressing scheduling problems. Their sophisticated reasoning capabilities enable comprehensive understanding of complex scheduling constraints and their intricate interactions. Furthermore, LLMs excel at learning

*Corresponding author

from historical experiences and identifying patterns, capabilities that are essential for developing effective heuristic search strategies. Their ability to process and interpret natural language descriptions of problems and objectives facilitates more flexible and adaptable algorithm design. These characteristics position LLMs as particularly promising tools for addressing challenging scheduling optimization problems.

Recent research has demonstrated the potential of Large Language Models in automating algorithm generation for combinatorial optimization problems [Romera-Paredes *et al.*, 2024; Liu *et al.*, 2024a; Liu *et al.*, ; Zeng *et al.*,]. However, current LLMs-based methodologies exhibit several critical limitations in addressing scheduling problems: (1) The predominant approach of generating complete algorithms directly [Romera-Paredes *et al.*, 2024; Liu *et al.*, 2024a; Liu *et al.*, ; Zeng *et al.*, ; Ye *et al.*, 2024] often proves inadequate for problems with complex constraints such as JSP, failing to maintain solution quality; (2) The inherent hallucination phenomenon in LLMs may produce unreliable or invalid outputs, potentially compromising algorithmic performance;

To address these limitations, we propose a novel LLMs-based neighborhood search framework that synergistically combines the strengths of LLMs and traditional local search methodologies. Instead of attempting to generate complete algorithms directly, our approach utilizes LLMs to guide neighborhood selection in local search, thereby maintaining solution feasibility while enhancing solution quality. Additionally, we introduce a verification evolutionary framework that systematically validates LLMs-generated suggestions and provides structured feedback, analogous to gradient information in deep learning, enabling progressive improvement in heuristic generation. Furthermore, the proposed LLMs-guided neighborhood search framework (NS4S) can be directly applied to other metaheuristics or even extended problems without manual parameter tuning, demonstrating strong generalization capability.

2 Literature Review

2.1 Heuristic Methods

Traditional methods for solving the JSP can be classified into heuristics and metaheuristics. Heuristics gradually construct a complete solution from an empty one that does not contain any operations, whereas metaheuristics usually perform improvement based on a complete feasible solution. In tackling the job shop scheduling problem, heuristics mostly rely on priority dispatching rules such as shortest processing time (SPT), first in first out (FIFO), most work remaining (MWKR) [Sels *et al.*, 2012], and so on. For JSP, prominent metaheuristic approaches include evolutionary algorithm [Pan *et al.*, 2021], particle swarm algorithm [Fontes *et al.*, 2023], and memetic algorithm [Sun *et al.*, 2023a]. In the context of FJSP, popular algorithms include the Jaya algorithm [Caldeira and Gnanavelbabu, 2019], hybrid search algorithm [Xie *et al.*, 2023], and genetic algorithm [Sun *et al.*, 2023b]. For solving FJSP-SDST, common strategies involve hybrid algorithm [Oddi *et al.*, 2011], tabu search [González *et al.*, 2013], and genetic algorithm [González *et al.*, 2013].

2.2 Learning-based Methods

With the development of machine learning theory, the methods based on deep learning have demonstrated their ability to tackle complex problems in the field of computer vision and combinatorial optimization. Chen *et al.* [2020] proposed a self-learning genetic algorithm by integrating reinforcement learning into genetic algorithms. Their algorithm can adaptively adjust the execution frequency of the Sarsa and Q-learning algorithms based on the features of the current search region to guarantee the quality of solutions. In order to reduce the reliance on expert knowledge for priority dispatching rules, Zhang *et al.* [2020] introduced an end-to-end deep reinforcement learning approach, which involves utilizing a graph neural network to extract features from incomplete solutions rather than commonly using complete feasible solutions. Finally, the policy network model outputs a dispatching rule suitable for the current state. Moreover, experimental results showed that their method also exhibits good generalization capabilities. Zhang *et al.* [2023] combined deep reinforcement learning with multi-agent systems to propose the deepMAG framework. In deepMAG, each agent has its own optimization objective. Additionally, neighboring agents can collaborate with each other by executing a shared action. Experimental results indicate that this collaborative approach can improve the algorithm's efficiency. Yuan *et al.* [2023] proposed a deep reinforcement learning-based method by utilizing graph-heterogeneous networks for feature embedding and modeled JSP as a standard Markov decision process.

2.3 Automatic Heuristic Design

Traditional automated heuristic design is commonly referred to as Hyper-heuristics, which typically involve the combination of various simple heuristic rules or select the best performing heuristic from a predefined set [Ye *et al.*, 2024]. In recent years, significant advancements have been made in natural language generation using LLMs. Consequently, some researchers have attempted to apply LLMs to generate entire algorithms and codes. DeepMind team [Romera-Paredes *et al.*, 2024] utilized a pre-trained large language model to generate heuristics for the cap set problem and the online packing problem, which led to new constructions in the cap set problem, breaking previous world records. In order to tackle black-box optimization problems, Liu *et al.* [2024c] embedded LLMs into a Bayesian optimization framework, enabling the LLMs to generate and evaluate feasible solutions based on context.

However, when the problem constraints exhibit high complexity, this direct generation of complete algorithms cannot guarantee the quality of solutions. Zhuang *et al.* [2024] employed LLMs to generate the cost function for specific tasks and integrated it with A* algorithm to effectively prune the search tree, achieving a trade-off between exploration and exploitation. Liu *et al.* [2024a] utilized LLMs to generate several heuristics for specific tasks and enhance the generated heuristics through selection, crossover, and mutation operations. To further improve the performance of heuristics, they performed parameter tuning and strategy optimization on the generated heuristics at different stages.

Algorithm 1 The main framework of the NS4S algorithm

Input: JSP Instance
Output: the best solution found so far S^*

```
1: S* ← S ← Init(),
2: f* ← S.makespan
3: while stopping condition is not met do
4:     move(o*, k) ← LLMGuidedNeighborEval(S)        /*Section 3.2*/
5:     S ⟵ S ⊕ move(o*, k)
6:     f ← S.makespan
7:     WeightUpdateByLLMs()        /*Section 3.3*/
8:     if f* > f then
9:         S* ← S, f* ← f
10:    end if
11: end while
12: Return S*
```

3 LLMs-guided Neighborhood Search

3.1 Overall Framework

In this section, we present a comprehensive framework that leverages LLMs to guide neighborhood search in solving classical scheduling problems. The framework's primary innovation lies in its utilization of LLMs to direct the neighborhood search process through dynamic weight adjustments, effectively steering the search towards promising regions of the solution space.

The main framework of our algorithm, detailed in Algorithm 1, comprises three fundamental components:

- An LLMs-guided neighborhood evaluation strategy that employs operation weights to assess move quality
- A verification-based evolutionary framework (VeEvo) that generates and refines high-quality weight adjustment heuristics
- A neighborhood search process that integrates these components to systematically explore the solution space

The algorithm begins by constructing an initial feasible solution through random assignment of operations to candidate machines with uniform probability (line 1). Solutions S^* and S represent the global best and current solutions respectively, with their corresponding objective values denoted by f^* and f (line 2). The algorithm then iteratively applies a series of moves to improve solution quality until meeting the termination criterion (lines 3-11).

Specifically, the proposed LLMs-guided neighborhood evaluation (LNE) strategy evaluates potential neighborhood moves of the incumbent solution to identify the most promising move (line 4). The selected move is then executed to obtain a new solution (line 5), where $move(o^*, k)$ denotes the relocation of operation o^* to position k. Subsequently, the weights of affected operations are updated based on the resulting changes in makespan (line 7), with the update heuristic generated by the proposed VeEvo framework. The best-found solution is updated when improvements are discovered (lines 8-10).

3.2 Neighborhood Structure and Evaluation

The design of neighborhood structure and evaluation strategy plays a pivotal role in determining the effectiveness of neighborhood search algorithms. While we adopt the established neighborhood structure from Ding *et al.* [2019], we propose a novel approach to perform move evaluation. Traditional local search methods typically evaluate all feasible neighborhood moves and select the one yielding the best evaluation value. Although this greedy strategy facilitates rapid convergence, it frequently leads to entrapment in local optima. To overcome this limitation, we introduce an LLMs-guided neighborhood evaluation (LNE) strategy that incorporates dynamically adjusted operation weights, learned through LLMs evolution, to guide the search process more effectively.

Consider a sequence of operations $O_1, O_2, \ldots, O_k$ processed on the same machine in JSP. When evaluating the insertion of operation O_1 after operation O_k, the evaluation metric is computed as follows (the move with the best F will be selected and performed at next iteration):

$$F = max\{R_i + p_i + Q_i + W(i)\}, \forall i \in \{O_1, ..., O_k\} \quad (1)$$

$$W(i) = CLIP(1 - t_i/rand(N), 1) \times w_i \quad (2)$$

Here, $W(i)$ represents the LLMs-guided weight component for operation i, p_i denotes its processing time, while R_i and Q_i indicate the earliest start time and the length of the longest path from operation i to the terminating operation prior to the move, respectively. The function $CLIP(x, 1)$ constrains values to the interval [0,1], t_i counts the iterations since operation i was last moved, and N represents the mean operation count across all jobs. To enhance exploration, $rand(N)$ generates a random integer in the range [1,N]. The weight w_i, initialized to 0, is dynamically updated using heuristics derived from our VeEvo framework.

This LLMs-guided evaluation strategy has several significant advantages: (1) It allows the search to escape from local optima by dynamically adjusting operation weights based on LLMs-generated heuristics. (2) The weights provide a form of learning that captures problem-specific knowledge through the search process. (3) The randomization factor helps to maintain diversity in the search process while still being guided by the learned weights.

According to Equations (1) and (2), the weights of operations can significantly impact move evaluation and influence the selection of neighboring moves. The VeEvo framework is employed to generate high-quality weight adjustment strategies that can effectively guide the search towards promising regions.

3.3 LLMs-based Weight Adjustment

In order to employ a robust mechanism for weight adjustment to effectively utilize LLMs for search guidance, we introduce a verification-based evolutionary framework (VeEvo) for generating high-quality weight adjustment strategies. A fundamental challenge in leveraging LLMs for this purpose stems from their susceptibility to hallucination [Xu *et al.*, 2024; Banerjee *et al.*, 2024; Liu *et al.*, 2024b; Li *et al.*, 2024], which

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

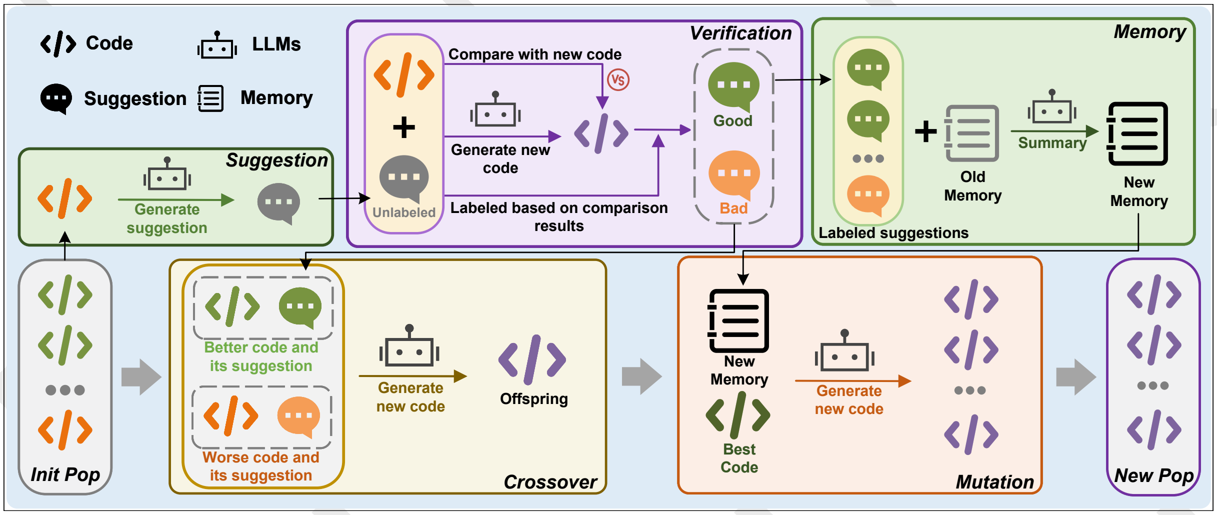


Figure 1: Schematic illustration of VeEvo.

can result in unreliable or counterproductive outputs. While existing LLMs-based evolutionary algorithms often assume output reliability, this assumption may lead to the exploration of unpromising search spaces. Our VeEvo framework addresses this challenge through a systematic approach to verification and evolutionary refinement.

The architecture of VeEvo is illustrated in Figure 1. The framework's effectiveness stems from three key mechanisms: (1) learning-based adaptation that enables continuous strategy refinement through evolutionary feedback, (2) systematic verification that ensures the reliability of LLMs-generated strategies through empirical validation, and (3) dynamic knowledge base optimization that facilitates continuous learning through systematic feedback mechanisms and comprehensive experience integration.

VeEvo adopts a methodology analogous to genetic programming in evolutionary computation. Following Romera *et al.* [2024], the framework evolves individual heuristic methods rather than problem solutions as in traditional genetic programming. The framework comprises the following essential components:

Initialization: The population is initialized through LLMs-generated heuristics based on carefully crafted prompts containing task descriptions, heuristic definitions, and illustrative examples. Generated heuristics undergo validation to eliminate invalid candidates, followed by evaluation and ranking based on their fitness values.

Suggestion Generation: The framework processes heuristic codes sequentially through the LLMs to generate natural language suggestions, which serve as gradient-like information to guide evolutionary crossover. To mitigate the impact of LLMs' uncertainty, suggestions for low-performing individuals (bottom 50% by fitness) are labeled as "Unlabeled" (e.g., "*# Unlabeled Suggestion: [Unlabeled] Increase weight for moved operations, decrease for unmoved with conditions favoring current optimization goal*"). These Unlabeled suggestions undergo additional verification.

Verification Process: Low-performing individuals and their associated suggestions are processed by the LLMs to generate improved heuristics. The new heuristics undergo evaluation and comparison with their predecessors. Superior performance results in a "Good suggestion" label, while inferior performance yields a "Bad suggestion" label (e.g., "*# Good Suggestion: [High Quality] Increase weight for moved operations...*"). These quality indicators inform subsequent evolutionary operations.

Crossover Operation: The LLMs generates offspring code based on parent codes, suggestions, and their corresponding quality labels. Rather than discarding ineffective suggestions, they are retained with "bad suggestion" labels, as they provide valuable negative gradient information for the learning process.

Memory Mechanism: VeEvo maintains an evolving knowledge base through continuous reflection on suggestion effectiveness. The framework accumulates expertise such as "Penalize stagnation and prioritize diverse solutions. Remember, non-negative weights are crucial for algorithm stability". The LLMs updates this knowledge base by processing new suggestions and their quality labels alongside existing memories, with "bad suggestion" labels reducing the acceptance probability of similar ineffective strategies.

Mutation Operation: The framework leverages the best-performing individual and accumulated memories to guide the LLMs in generating diverse variants. These new individuals, combined with crossover-generated offspring, form the population of the next generation.

During training, VeEvo iteratively executes these processes to enhance population quality, using average objective values across randomly generated instances as fitness metrics. In the testing phase, the best-evolved heuristic is integrated directly into Algorithm 1 (Line 7), eliminating the need for further LLMs interaction during execution.

4 Experiments and Analysis

4.1 Experimental Protocol

Benchmark Datasets: To evaluate the efficacy of the proposed NS4S algorithm, we conducted comprehensive experiments across three classical scheduling problems: JSP,

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

Algorithm		Size								Avg
		15×15	20×15	20×20	30×15	30×20	50×15	50×20	100×20	
SPT	Obj	1546.1	1813.5	2067.0	2419.3	2619.1	3441.0	3570.8	6139.0	2952.0
	Gap	25.89%	32.82%	27.75%	35.27%	34.44%	24.11%	25.54%	14.41%	27.53%
FDD/WKR	Obj	1808.6	2054.0	2387.2	2590.8	3045.0	3736.3	4022.1	6620.7	3283.1
	Gap	47.15%	50.57%	47.61%	45.02%	56.30%	34.77%	41.50%	23.39%	43.29%
MWKR	Obj	1464.3	1683.6	1969.8	2214.8	2439.0	3240.0	3352.8	5812.2	2772.1
	Gap	19.15%	23.35%	21.81%	23.91%	25.17%	16.86%	17.95%	8.31%	19.56%
MOPNR	Obj	1481.3	1686.7	1968.3	2195.8	2433.6	3254.5	3346.9	5856.9	2778.0
	Gap	20.53%	23.55%	21.71%	22.83%	24.94%	17.37%	17.68%	9.15%	19.72%
L2D	Obj	1547.4	1774.7	2128.1	2378.8	2603.9	3393.8	3593.9	6097.6	2939.8
	Gap	25.96%	30.03%	31.61%	33.00%	33.62%	22.38%	26.51%	13.61%	27.09%
RASCL	Obj	1339.8	1509.3	1793.1	2038.1	2261.5	3030.8	3125.1	5578.9	2584.6
	Gap	9.02%	10.58%	10.87%	13.98%	16.09%	9.32%	9.89%	3.96%	10.46%
EYRL	Obj	1447.8	1645.8	1933.2	2189.1	2403.2	3217.3	3338.2	5845.0	2752.5
	Gap	17.85%	20.59%	19.53%	22.39%	23.32%	16.03%	17.41%	8.91%	18.25%
NS4S(GPT3.5)	Obj	1239.6	1389.5	1648.6	1820.0	2025.6	**2773.8**	2855.0	**5443.3**	2399.4
	Gap	0.87%	1.81%	1.94%	1.70%	3.98%	**0.00%**	0.39%	**1.45%**	1.52%
NS4S(GPT4)	Obj	**1239.0**	**1388.2**	**1647.1**	**1814.9**	**2012.6**	**2773.8**	**2849.2**	5446.2	**2396.4**
	Gap	**0.82%**	**1.70%**	**1.85%**	**1.41%**	**3.31%**	**0.00%**	**0.19%**	1.50%	**1.35%**

Table 1: Summary of results on the TA benchmark in JSP.

Algorithm		Size								Avg
		20×15	20×20	30×15	30×20	40×15	40×20	50×15	50×20	
SPT	Obj	4951.5	5690.5	6306.2	7036.0	7601.2	8538.1	8975.4	10132.8	7404.0
	Gap	64.13%	64.57%	62.57%	65.91%	55.88%	63.00%	50.37%	62.20%	61.08%
FDD/WKR	Obj	4666.3	5298.2	6016.5	6827.3	7420.0	8210.9	9150.2	9899.6	7186.1
	Gap	53.57%	52.52%	54.12%	60.09%	51.42%	55.52%	52.53%	57.26%	54.63%
MWKR	Obj	4909.9	5489.0	6252.9	6925.0	7484.2	8460.9	8906.0	9807.0	7279.4
	Gap	62.15%	58.16%	60.95%	63.16%	52.87%	61.11%	48.93%	56.40%	57.97%
MOPNR	Obj	4513.2	5052.3	5742.8	6491.9	7105.5	7870.7	8436.5	9408.0	6827.6
	Gap	49.16%	45.17%	47.14%	51.97%	44.72%	49.22%	40.79%	49.61%	47.22%
L2D	Obj	4215.3	4804.5	5557.9	5967.4	6663.9	7375.8	8179.4	8751.6	6439.5
	Gap	38.95%	37.74%	41.86%	39.48%	35.38%	39.38%	36.20%	38.86%	38.48%
RASCL	Obj	3610.0	4028.9	4522.0	5106.0	5731.9	6584.1	7242.1	7176.9	5500.2
	Gap	19.36%	15.98%	16.35%	20.00%	17.49%	25.42%	21.54%	14.66%	18.85%
EYRL	Obj	3839.7	4332.8	5012.5	5524.7	6108.8	6858.6	7479.1	8150.7	5913.4
	Gap	26.56%	23.55%	28.69%	29.21%	24.46%	30.06%	24.93%	29.51%	27.12%
NS4S(GPT3.5)	Obj	3117.3	3587.9	4077.9	4519.8	5120.1	5608.0	6313.4	6735.5	4885.0
	Gap	2.89%	3.17%	4.63%	5.86%	4.61%	6.36%	5.63%	7.09%	5.03%
NS4S(GPT4)	Obj	**3106.9**	**3579.5**	**4047.9**	**4499.0**	**5099.2**	**5553.2**	**6288.3**	**6721.5**	**4861.9**
	Gap	**2.57%**	**2.96%**	**3.91%**	**5.37%**	**4.21%**	**5.33%**	**5.23%**	**6.84%**	**4.55%**

Table 2: Summary of results on the DMU benchmark in JSP.

FJSP, and FJSP-SDST. For JSP, we utilized the TA and DMU benchmark instances, comprising 160 instances that represent the most challenging publicly available test cases. For FJSP, we employed the BR, HU, and Vdata benchmark suites, encompassing 169 diverse instances. For FJSP-SDST, we employed the SDST-Hudata benchmark set, containing 20 instances of varying complexity.

Baseline Algorithms: Our comparative analysis encompasses three categories of baseline algorithms:

1) Priority Dispatch Rules (PDRs): Following Sels *et al.* [2012], we selected widely-adopted heuristics including Shortest Processing Time (SPT), Minimum Ratio of Flow Due Date to Most Work Remaining (FDD/WKR), Most Work Remaining (MWKR), Most Operations Remaining (MOPNR), First In First Out (FIFO), and machine with Earliest End Time (EET).

2) State-of-the-art Metaheuristics: We incorporated advanced algorithms including Improved Jaya Algorithm (IJA) [Caldeira and Gnanavelbabu, 2019], Regular GA (RegGA) [Rooyani and Defersha, 2019], Two Stage Genetic Algorithm (2SGA) [Rooyani and Defersha, 2019], Self-learning Genetic Algorithm (SLGA) [Chen *et al.*, 2020], Iterative Flat-

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

Algorithm	Barnes		Brandimarte		Dauzere		Hurink-rdata		Hurink-edata		Hurink-vdata	
	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)
IJA	2.40%	21.03	8.50%	15.43	6.10%	180.8	3.90%	19.72	4.60%	15.24	2.70%	17.85
RegGA	–	–	8.39%	280.1	–	–	–	–	–	–	3.20%	191.4
2SGA	–	–	3.17%	57.6	–	–	–	–	–	–	0.39%	51.43
SLGA	–	–	6.21%	283.28	–	–	–	–	–	–	–	–
FIFO+EET	27.91%	0.019	28.98%	0.017	15.00%	0.036	17.38%	0.018	19.89%	0.016	7.14%	0.019
MWKR+EET	54.71%	0.018	39.50%	0.019	24.69%	0.036	26.60%	0.018	44.68%	0.018	8.96%	0.02
MOPNR+EET	50.66%	0.017	43.39%	0.018	32.17%	0.038	26.47%	0.019	43.67%	0.018	13.14%	0.019
EYRL	13.83%	0.391	13.24%	0.406	11.02%	0.808	12.09%	0.275	15.54%	0.271	5.37%	0.272
WSRL	17.88%	1.516	30.04%	1.305	8.88%	2.716	11.02%	1.421	16.66%	1.424	4.41%	1.405
LKRL	28.96%	2.048	13.59%	2.054	15.68%	3.917	16.51%	1.591	23.01%	1.558	6.96%	1.544
NS4S(GPT3.5)	1.26%	1.016	0.17%	1.027	0.66%	3.513	0.14%	1.704	0.07%	1.074	0.10%	1.154
NS4S(GPT4)	**1.20%**	**2.026**	**0.05%**	**1.455**	**0.31%**	**5.117**	**0.00%**	**2.412**	**0.01%**	**1.448**	**0.09%**	**2.609**

Table 3: Summary of results on the FJSP benchmarks.

tening Search (IFS) [Oddi *et al.*, 2011], Tabu Search (TS) [González *et al.*, 2013], Genetic Algorithm (GA) [González *et al.*, 2013], and Memetic Algorithm (MA) [González *et al.*, 2013].

3) Learning-based Methods: We included contemporary approaches such as Learning to Dispatch (L2D) [Zhang *et al.*, 2020], Reinforced Adaptive Staircase Curriculum Learning (RASCL) [Iklassov *et al.*, 2023], and deep reinforcement learning-based methods from [Yuan *et al.*, 2024; Song *et al.*, 2022; Lei *et al.*, 2022], denoted as EYRL, WSRL, and LKRL, respectively.

Experimental Settings: Our implementation combines Python for the VeEvo framework and C++ for the neighborhood search algorithm, executed on a single CPU E5-2697v3. To ensure fair comparison with Ye *et al.* [2024], we configured VeEvo with a population size of 10 and the maximum evaluations of 100. We imposed cutoff times of 10 seconds for JSP and FJSP, and 30 seconds for FJSP-SDST.

To mitigate implementation-dependent variations, we adopted the computer-independent CPU time normalization methodology from Sels *et al.* [2024]. Reference results for JSP were cited from Iklassov *et al.* [2023] and Yuan *et al.* [2023], while FJSP and FJSP-SDST results were obtained from Yuan *et al.* [2024] and Gonzalez *et al.* [2013].

For performance evaluation, we employed the relative percentage deviation (RPD), also known as the optimality gap, following Yuan *et al.* [2023]:

$$RPD = (C^{'}_{max} - C^{*}_{max})/C^{*}_{max} \times 100\% \quad (3)$$

where $C^{'}_{max}$ represents the best makespan achieved by the reference algorithm, and C^{*}_{max} denotes the optimal or best-known solution. In the tables, the best performance across all algorithms are indicated in bold, while asterisks (*) denote new upper bounds established by our approach.

4.2 Experimental Results

JSP Performance Analysis: As evidenced in Table 1, our approach demonstrates substantial improvement over existing methods. While RASCL achieves a notable average gap of 10.46% on the TA instances, surpassing other rule-based heuristics and deep learning methods, NS4S significantly reduces this gap to 1.35%. This improvement indicates the effectiveness of our LLMs-based neighborhood approach in navigating complex solution spaces.

The integration of LLMs capabilities proves particularly effective, as demonstrated by the performance on DMU instances (Table 2). The GPT-4-based variant achieves an average gap of 4.55%, substantially outperforming RASCL's 18.85%. Notably, this improvement is achieved without explicit weight update direction instructions in the initial prompts, highlighting VeEvo's ability to learn effective strategies through self-verification and evolutionary refinement.

FJSP Performance Analysis: Results presented in Table 3 reveal the balanced performance characteristics of our approach. While genetic programming-based algorithms achieve high solution quality at the cost of substantial computation time, and rule-based heuristics offer rapid but suboptimal solutions, NS4S strikes an optimal balance. It outperforms reinforcement learning approaches like EYRL, WSRL, and LKRL in terms of solution quality while maintaining comparable computational efficiency. The GPT-4 variant demonstrates superior solution quality across all benchmark sets, though with moderately increased computation time.

FJSP-SDST Performance Analysis: The results in Table 4 underscore the robustness of our approach in handling complex constraints. NS4S not only surpasses the Memetic Algorithm (MA) in terms of both solution quality and computational efficiency but also establishes nine new upper bounds across the benchmark instances[1]. Specifically, the GPT-3.5 variant established records on instances la06, la07, la09, la10, and la12, while the GPT-4 variant achieved breakthroughs on la06, la09, la11, la13, la14, and la15.

4.3 Analysis and Discussion

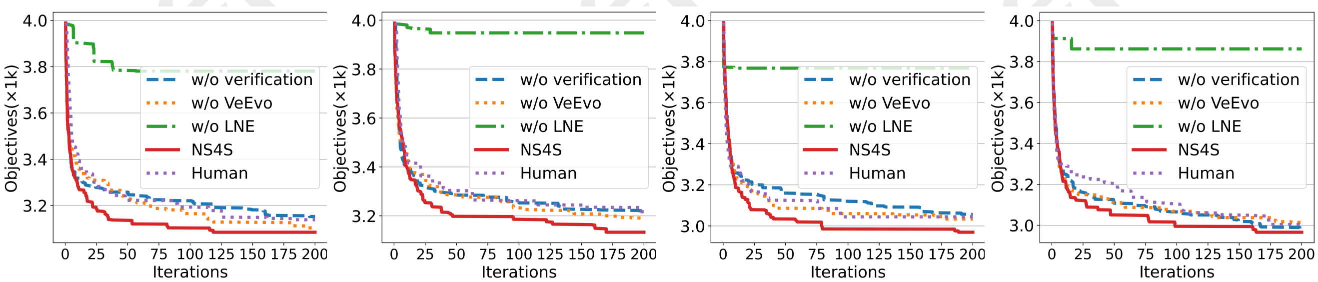
To evaluate the importance of each algorithmic component, we conducted ablation studies on the four most challenging instances from the SVW benchmark. We compared

[1]https://github.com/Zoommy/NS4S

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

Ins.	LB	IFS	GA		TS		MA			NS4S(GPT3.5)			NS4S(GPT4)		
		Best	Best	Avg	Best	Avg	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
la01	609	726	801	817	721	724	721	724	6	**721**	721.0	2.16	**721**	721.5	9.87
la02	655	749	847	870	737	738	737	737	7	**737**	737.5	7.16	**737**	737.5	9.62
la03	550	652	760	789	652	652	652	652	7	**652**	652.0	0.40	**652**	652.0	0.37
la04	568	673	770	790	673	678	673	675	9	**673**	673.0	1.04	**673**	673.0	1.71
la05	503	603	679	685	602	602	602	602	8	**602**	602.0	1.76	**602**	602.0	0.84
la06	833	950	1147	1165	956	961	953	957	12	**945***	946.2	10.66	**945***	946.5	9.84
la07	762	916	1123	1150	912	917	905	911	18	**902***	905.6	9.10	904	907.9	12.92
la08	845	948	1167	1186	940	951	940	941	15	**940**	942.4	14.32	**940**	941.4	13.02
la09	878	1002	1183	1210	1002	1007	989	995	22	**984***	988.2	9.19	**984***	986.0	14.36
la10	866	977	1127	1156	956	960	956	956	29	**953***	955.4	3.39	956	956.0	0.61
la11	1087	1256	1577	1600	1265	1273	1244	1254	33	1236	1239.9	9.32	**1232***	1235.6	8.93
la12	960	1082	1365	1406	1105	1119	1098	1107	26	**1070***	1081.2	12.52	1077	1081.9	8.11
la13	1053	1215	1473	1513	1210	1223	1205	1212	24	1180	1182.6	15.73	**1172***	1177.6	18.29
la14	1123	1285	1549	1561	1367	1277	1257	1263	27	1237	1244.6	12.15	**1234***	1242.4	15.49
la15	1111	1291	1649	1718	1284	1297	1275	1282	29	1261	1266.9	10.49	**1258***	1262.2	17.12
la16	892	1007	1256	1269	1007	1007	1007	1007	12	**1007**	1007.0	0.29	**1007**	1007.0	0.46
la17	707	858	1007	1059	851	851	851	851	12	**851**	851.0	2.79	**851**	854.0	2.37
la18	842	985	1146	1184	985	988	985	992	10	**985**	991.4	4.21	**985**	987.4	14.33
la19	796	956	1166	1197	951	955	951	951	16	**951**	953.0	11.81	**951**	951.5	9.07
la20	857	997	1194	1228	997	997	997	997	12	**997**	997.0	1.01	**997**	997.0	1.37

Table 4: Summary of results on SDST-HUdata benchmark in FJSP-SDST.



(a) SWV12 (b) SWV13 (c) SWV14 (d) SWV15

Figure 2: Evolution of the makespan by NS4S and others on four hardest instances in *SWV* benchmark.

NS4S against variants lacking specific components: verification mechanism (w/o verification), LLMs-guided neighborhood evaluation (w/o LNE), VeEvo framework (replaced with ReEvo [Ye *et al.*, 2024]), and expert-designed weight adjustment heuristics (Human version).

As illustrated in Figure 2, the variant without LNE exhibits significantly slower improvement rates, highlighting the crucial role of LLMs-guided neighborhood evaluation in efficient solution space exploration. While the variants without verification, without VeEvo, and the Human version show similar convergence patterns, they consistently achieve inferior solutions compared to the complete NS4S framework. The full NS4S implementation's ability to obtain superior solutions with reduced computational time validates the synergistic benefits of integrating these components.

5 Conclusion

This paper presents a novel LLMs-based neighborhood search method that leverages Large Language Models to address classical scheduling problems including JSP, FJSP, and FJSP-SDST. Our primary contributions encompass three key innovations: First, we introduce an LLMs-guided neighborhood evaluation strategy that employs dynamic weight adjustments to effectively guide the search process through complex solution spaces. Second, we develop a verification-based evolutionary framework (VeEvo) that systematically mitigates the impact of LLMs hallucinations through rigorous validation of generated heuristics. Third, we demonstrate how the integration of LLMs-generated weight adjustment strategies can effectively steer the search towards promising regions of the solution space.

Comprehensive experimental evaluation across three classical scheduling problems, encompassing 349 benchmark instances, demonstrates the superiority of our approach over existing state-of-the-art methods. For the Job Shop Scheduling Problem, our algorithm achieves a remarkable reduction in average optimality gap from 10.46% to 1.35% on Taillard's instances. In the context of FJSP, we further reduce the average optimality gap from 13.24% to 0.05% on Brandimarte's instances. Most notably, for the FJSP-SDST, our approach establishes nine new upper bounds.

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62472189, 62402191, and 62202192, the Special Project for Knowledge Innovation of Hubei Province under Grant 2022013301015175, and Interdiciplinary Research Program of Hust 5003300129.

References

[Banerjee *et al.*, 2024] Sourav Banerjee, Ayushi Agarwal, and Saloni Singla. Llms will always hallucinate, and we need to live with this. *arXiv preprint arXiv:2409.05746*, 2024.

[Caldeira and Gnanavelbabu, 2019] Rylan H. Caldeira and A. Gnanavelbabu. Solving the flexible job shop scheduling problem using an improved jaya algorithm. *Computers & Industrial Engineering*, 137:106064, 2019.

[Chen *et al.*, 2020] Ronghua Chen, Bo Yang, Shi Li, and Shilong Wang. A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Computers & industrial engineering*, 149:106778, 2020.

[Ding *et al.*, 2019] Junwen Ding, Zhipeng Lü, Chu-Min Li, Liji Shen, Liping Xu, and Fred Glover. A two-individual based evolutionary algorithm for the flexible job shop scheduling problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2262–2271, 2019.

[Fontes *et al.*, 2023] Dalila BMM Fontes, S Mahdi Homayouni, and José F Gonçalves. 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, 2023.

[González *et al.*, 2013] Miguel Ángel González, Camino Rodríguez Vela, and Ramiro Varela. An efficient memetic algorithm for the flexible job shop with setup times. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 23, pages 91–99, 2013.

[Guo *et al.*, 2024] Haoxin Guo, Jianhua Liu, Yue Wang, and Cunbo Zhuang. An improved genetic programming hyperheuristic for the dynamic flexible job shop scheduling problem with reconfigurable manufacturing cells. *Journal of Manufacturing Systems*, 74:252–263, 2024.

[Iklassov *et al.*, 2023] Zangir Iklassov, Dmitrii Medvedev, Ruben Solozabal Ochoa De Retana, and Martin Takac. On the study of curriculum learning for inferring dispatching policies on the job shop scheduling. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI*, pages 5350–5358, 2023.

[Lei *et al.*, 2022] Kun Lei, Peng Guo, Wenchao Zhao, Yi Wang, Linmao Qian, Xiangyin Meng, and Liansheng Tang. A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem. *Expert Systems with Applications*, 205:117796, 2022.

[Li *et al.*, 2024] Junyi Li, Jie Chen, Ruiyang Ren, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. The dawn after the dark: An empirical study on factuality hallucination in large language models. *arXiv preprint arXiv:2401.03205*, 2024.

[Liu *et al.*,] Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *The Twelfth International Conference on Learning Representations*.

[Liu *et al.*, 2024a] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language mode. *arXiv preprint arXiv:2401.02051*, 2024.

[Liu *et al.*, 2024b] Hanchao Liu, Wenyuan Xue, Yifei Chen, Dapeng Chen, Xiutian Zhao, Ke Wang, Liping Hou, Rongjun Li, and Wei Peng. A survey on hallucination in large vision-language models. *arXiv preprint arXiv:2402.00253*, 2024.

[Liu *et al.*, 2024c] Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *The Twelfth International Conference on Learning Representations*, 2024.

[Oddi *et al.*, 2011] Angelo Oddi, Riccardo Rasconi, Amedeo Cesta, and Stephen F Smith. Job shop scheduling with routing flexibility and sequence dependent setup-times. 2011.

[Pan *et al.*, 2021] Zixiao Pan, Deming Lei, and Ling Wang. A bi-population evolutionary algorithm with feedback for energy-efficient fuzzy flexible job shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(8):5295–5307, 2021.

[Romera-Paredes *et al.*, 2024] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[Rooyani and Defersha, 2019] Danial Rooyani and Fantahun M Defersha. An efficient two-stage genetic algorithm for flexible job-shop scheduling. *IFAC-PapersOnLine*, 52(13):2519–2524, 2019.

[Sels *et al.*, 2012] Veronique Sels, Nele Gheysen, and Mario Vanhoucke. A comparison of priority rules for the job shop scheduling problem under different flow time-and tardiness-related objective functions. *International Journal of Production Research*, 50(15):4255–4270, 2012.

[Song *et al.*, 2022] 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, 2022.

[Sun *et al.*, 2023a] Kexin Sun, Debin Zheng, Haohao Song, Zhiwen Cheng, Xudong Lang, Weidong Yuan, and Jiquan

Preprint – IJCAI 2025: This is the accepted version made available for conference attendees.
Do not cite. The final version will appear in the IJCAI 2025 proceedings.

Wang. Hybrid genetic algorithm with variable neighborhood search for flexible job shop scheduling problem in a machining system. *Expert Systems with Applications*, 215:119359, 2023.

[Sun *et al.*, 2023b] Kexin Sun, Debin Zheng, Haohao Song, Zhiwen Cheng, Xudong Lang, Weidong Yuan, and Jiquan Wang. Hybrid genetic algorithm with variable neighborhood search for flexible job shop scheduling problem in a machining system. *Expert Systems with Applications*, 215:119359, 2023.

[Tian *et al.*, 2024] Shichen Tian, Chunjiang Zhang, Jiaxin Fan, Xinyu Li, and Liang Gao. A genetic algorithm with critical path-based variable neighborhood search for distributed assembly job shop scheduling problem. *Swarm and Evolutionary Computation*, 85:101485, 2024.

[Xie *et al.*, 2023] Jin Xie, Xinyu Li, Liang Gao, and Lin Gui. A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems. *Journal of Manufacturing Systems*, 71:82–94, 2023.

[Xu *et al.*, 2024] Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. Hallucination is inevitable: An innate limitation of large language models. *arXiv preprint arXiv:2401.11817*, 2024.

[Yao *et al.*, 2024] Youjie Yao, Lin Gui, Xinyu Li, and Liang Gao. Tabu search based on novel neighborhood structures for solving job shop scheduling problem integrating finite transportation resources. *Robotics and Computer-Integrated Manufacturing*, 89:102782, 2024.

[Ye *et al.*, 2024] Haoran Ye, Jiarui Wang, Zhiguang Cao, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *arXiv preprint arXiv:2402.01145*, 2024.

[Yuan *et al.*, 2023] Erdong Yuan, Shuli Cheng, Liejun Wang, Shiji Song, and Fang Wu. Solving job shop scheduling problems via deep reinforcement learning. *Applied Soft Computing*, 143:110436, 2023.

[Yuan *et al.*, 2024] Erdong Yuan, Liejun Wang, Shuli Cheng, Shiji Song, Wei Fan, and Yongming Li. Solving flexible job shop scheduling problems via deep reinforcement learning. *Expert Systems with Applications*, 245:123019, 2024.

[Zeng *et al.*,] Junhua Zeng, Chao Li, Zhun Sun, Qibin Zhao, and Guoxu Zhou. tngps: Discovering unknown tensor network structure search algorithms via large language models (llms). In *Forty-first International Conference on Machine Learning*.

[Zhang *et al.*, 2020] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in neural information processing systems*, 33:1621–1632, 2020.

[Zhang *et al.*, 2023] Jia-Dong Zhang, Zhixiang He, Wing-Ho Chan, and Chi-Yin Chow. Deepmag: Deep reinforcement learning with multi-agent graphs for flexible job shop scheduling. *Knowledge-Based Systems*, 259:110083, 2023.

[Zhang *et al.*, 2024] Junjie Zhang, Zhipeng Lü, Junwen Ding, Zhouxing Su, Xingyu Li, and Liang Gao. An effective local search algorithm for flexible job shop scheduling in intelligent manufacturing systems. *Engineering*, 2024.

[Zhuang *et al.*, 2024] Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. Toolchain*: Efficient action space navigation in large language models with a* search. In *The Twelfth International Conference on Learning Representations*, 2024.