International Journal of Applied Operational Research
Vol. 13, No. 3, pp. 61-74, Summer 2025

Journal homepage: ijorlu.liau.ac.ir

A fast and scalable heuristic for makespan minimization in permutation flowshop scheduling

A. Olalekan Olasupo*, E. Olasunkanmi, O. Ogunfuye

Received: 28 March 2025 ; **Accepted:** 24 June 2025

Abstract The permutation flowshop scheduling problem (PFSP) is a classical NP-hard problem in production and operations management, where the objective is to minimize the makespan across multiple machines. Although established heuristics such as NEH, Gupta, and CDS are widely applied, their performance often declines in large-scale instances due to increased computational time and reduced scalability. This study proposes a fast heuristic based on a modified Johnson's rule applied pairwise between the first machine and each subsequent machine. For each pair, Johnson's two-machine algorithm generates a sequence, which is then evaluated on the full set of machines, and the best-performing sequence is selected as the final solution. Computational experiments on randomly generated instances of different sizes demonstrate that the proposed method achieves competitive makespan performance while significantly reducing CPU time compared to NEH and CDS, and providing better scalability than Gupta. Statistical validation using the Wilcoxon signed-rank test confirms that the proposed heuristic outperforms Gupta in solution quality and is considerably faster than NEH and CDS in execution time. These findings establish the proposed heuristic as a computationally efficient and statistically reliable approach for solving large-scale PFSPs, providing a valuable tool for production scheduling in industrial operations.

Keyword: Permutation Flow shop Scheduling, Makespan Minimization, Heuristic, Johnson's Algorithm, Scalability.

1 Introduction

Scheduling is one of the most critical decision-making areas in production and operations management, playing a vital role in improving efficiency and productivity in industrial systems. Among various scheduling problems, the permutation flowshop scheduling problem (PFSP) has received considerable attention because of its direct application in manufacturing and service industries. In this problem, a set of jobs must be processed on a series of machines in the same order, with the primary objective of minimizing the makespan the total completion

* **Corresponding Author. (✉)**
E-mail: olasupoolalekanazeez222@gmail.com (A. Olalekan Olasupo)

A. Olalekan Olasupo
Department of Industrial and Production Engineering, University of Ibadan, Nigeria.

E. Olasunkanmi
Department of Industrial and Production Engineering, University of Ibadan, Nigeria.

O. Ogunfuye
Department of Industrial and Production Engineering, University of Ibadan, Nigeria.

time of all jobs. Since the PFSP is an NP-hard problem when the number of machines is three or more, finding the optimal solution becomes computationally infeasible for large instances, which has led to the development of numerous heuristic and metaheuristic approaches [1,2].

Several heuristics have been proposed to solve the PFSP efficiently. Johnson's algorithm remains the earliest and most fundamental, providing an optimal solution for two-machine systems [3]. Later extensions such as Gupta's heuristic [4] and the Campbell–Dudek–Smith (CDS) method [5] generalized Johnson's rule for more machines, though sometimes at the cost of solution quality. The NEH heuristic, introduced later, remains one of the most effective constructive algorithms in terms of makespan performance and has been widely adopted in both literature and practice [6]. However, NEH's high computational cost limits its scalability, particularly for large problem sizes [7].

As production systems grow increasingly complex, industries require scheduling algorithms that can deliver high-quality solutions in shorter computational time. Existing heuristics often present trade-offs between accuracy and computational speed. Johnson's rule is computationally efficient but limited to two-machine systems, while Gupta and CDS methods extend its logic but tend to yield less accurate results. On the other hand, NEH consistently produces superior makespan performance but is time-intensive, making it less practical for large-scale or real-time applications [8,9].

To overcome these limitations, researchers have developed hybrid and modified heuristics aimed at improving computational efficiency without compromising solution quality. Approaches such as genetic algorithms [10], ant-colony optimization [11], and particle swarm optimization [12] have demonstrated promising results in reducing makespan while maintaining scalability. Despite these advances, there remains a need for simple, fast, and easily implementable heuristics that maintain strong performance across different problem sizes [13,14].

This study addresses this gap by developing a fast and scalable heuristic for the PFSP based on a modified Johnson's rule applied pairwise between the first machine and each subsequent machine. The algorithm generates multiple sequences and selects the best-performing one according to total makespan [15]. The proposed approach is evaluated against classical heuristics such as NEH, Gupta, and CDS across various problem sizes. The results demonstrate that the proposed method achieves competitive makespan performance while significantly reducing CPU time, thereby offering a practical solution for large-scale scheduling environments.

The remainder of this paper is organized as follows: Section 2 reviews related literature, Section 3 presents the proposed heuristic, Section 4 describes the experimental setup and comparative results, and Section 5 concludes the study with key findings and future recommendations.

2 Objective and challenge of PFSP

The permutation flowshop scheduling problem (PFSP) is one of the most critical and well-studied problems in operations research and production management. Its objective is to determine the sequence of jobs on a set of machines such that the makespan the total time required to complete all jobs is minimized [16,17]. The problem becomes computationally challenging when the number of machines exceeds two because PFSP is classified as NP-hard, meaning that exact optimization approaches require exponential time for large instances, which is impractical for real-world applications. Large-scale manufacturing systems, where hundreds

or thousands of jobs need to be processed on multiple machines under strict production deadlines, exemplify the scenarios where efficient heuristic or metaheuristic approaches are essential. Beyond makespan minimization, PFSP variants often consider additional practical constraints such as sequence-dependent setup times, machine availability, blocking, and no-wait conditions, which further increase the complexity of finding feasible and efficient schedules. Therefore, developing heuristics that are both computationally efficient and capable of producing high-quality solutions for a wide range of problem sizes has been a central focus in the literature [16,17].

The challenge of PFSP is not only computational but also practical. In many industrial systems, scheduling decisions must be made rapidly, sometimes in real-time, to respond to dynamic job arrivals, machine breakdowns, or changes in order priorities. Classical exact methods, such as branch and bound or dynamic programming, are often infeasible for such applications. Consequently, the research focus has shifted to constructive heuristics, metaheuristics, and hybrid approaches that strike a balance between solution quality and computational efficiency [16,17]. Additionally, contemporary manufacturing environments, characterized by mass customization and high variability, require heuristics that are flexible, scalable, and robust to uncertainty. Addressing these challenges has motivated the continuous evolution of PFSP solution approaches over the past several decades.

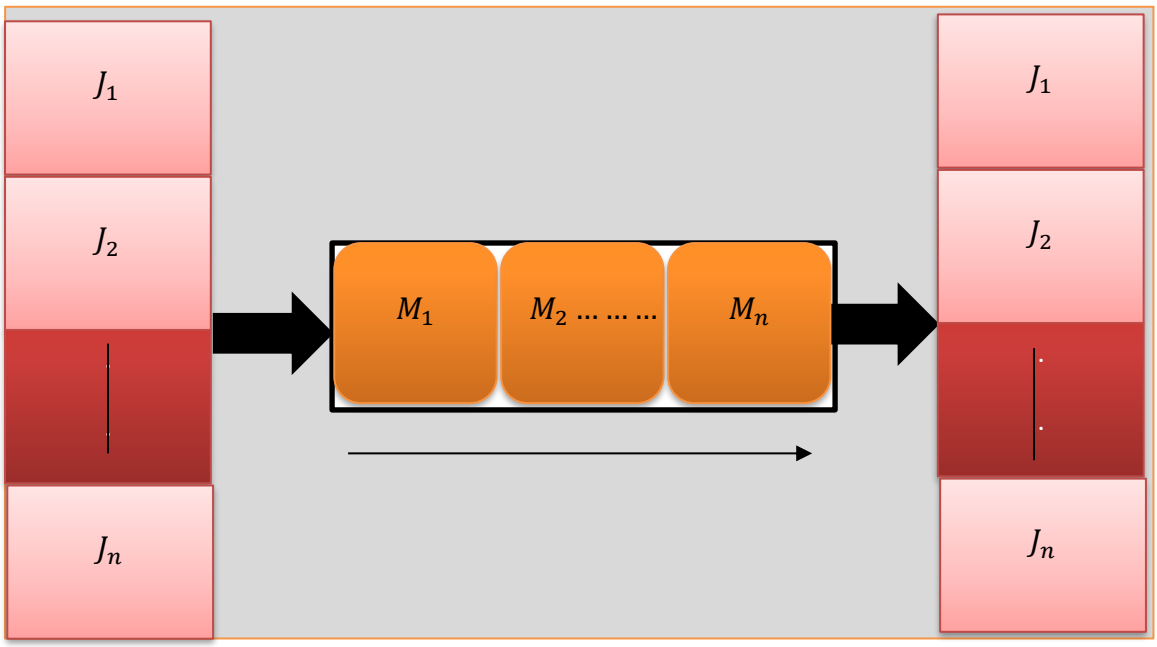

Fig. 1 Permutation Flow Shop Problem Diagram (J represent Jobs and M represent machine).

2.1 Classical and Constructive Heuristics

Classical heuristics form the foundation for solving PFSP efficiently. Johnson's rule, introduced in 1954, provides an optimal solution for two-machine flowshop problems by sequencing jobs based on the shortest processing times on the first and second machines [18]. The simplicity and optimality of Johnson's method make it extremely efficient for small-scale problems,

requiring minimal computational resources. However, Johnson's rule is inherently limited to two-machine cases and cannot be directly applied to multi-machine environments, which are more representative of practical industrial systems. To address this limitation, several extensions and generalizations have been proposed.

Gupta [19] introduced a heuristic that generalizes Johnson's principle to multi-machine systems by partitioning jobs into two groups based on their processing times and ordering them according to an index derived from sums of adjacent machine processing times. While Gupta's heuristic is computationally fast and effective for small to medium-sized problems, it does not account for interactions between non-adjacent machines and tends to produce suboptimal makespan in larger instances. The Campbell–Dudek–Smith (CDS) heuristic further extends the idea by aggregating multi-machine problems into multiple two-machine subproblems and applying Johnson's rule to each subproblem, generating a set of candidate sequences and selecting the best one [20]. Although CDS improves solution quality over Gupta in some cases, aggregation can obscure critical inter-machine interactions and may lead to higher makespan for complex systems.

The NEH heuristic, proposed by Nawaz et al. [21], iteratively constructs a sequence by inserting jobs at positions that minimize the partial makespan. NEH has been widely regarded as one of the most effective constructive methods for PFSP due to its consistent performance across a variety of problem sizes and configurations. However, its computational complexity increases rapidly with the number of jobs, and the performance can be sensitive to tie-breaking rules during insertion. Several enhancements to NEH have been proposed to mitigate these limitations. For instance, some studies introduced restricted insertion windows or reduced candidate sets to limit the number of sequences evaluated at each step [22,23,24,25]. Other approaches combine NEH with local search or neighborhood reduction heuristics to improve solution quality while reducing computation time [26,27]. These studies collectively highlight the fundamental trade-off between runtime efficiency and solution quality in classical heuristics: simpler methods are faster but may produce lower-quality solutions, whereas more sophisticated constructive methods achieve better makespan at the cost of higher computational effort.

2.2 Hybrid, Metaheuristic, and Learning-Based Approaches

In response to the limitations of classical heuristics, hybrid and metaheuristic strategies have been extensively explored in recent decades. Iterated greedy (IG) methods refine initial sequences, often generated by NEH, by iteratively destructing and reconstructing parts of the solution to escape local optima [28, 29]. Genetic algorithms (GA) and differential evolution methods leverage population-based search strategies to explore large solution spaces and identify near-optimal sequences, adapting operators such as crossover and mutation to the PFSP context [30, 31]. Tabu search (TS) enhances solution quality by using memory structures to avoid cycling and guide the search towards unexplored regions of the solution space [32, 33], while simulated annealing (SA) applies probabilistic acceptance criteria to overcome local minima [34].

Recently, integration of machine learning techniques has emerged as a promising avenue to improve PFSP solution efficiency. Reinforcement learning and Q-learning have been used to dynamically guide job insertion decisions and prune unpromising sequences [35, 36]. Guo (2024) demonstrated that a Q-learning hybrid with NEH adaptively adjusts insertion strategies to achieve lower makespan than conventional NEH, although with additional computational

overhead [37]. Zhou et al. [38] applied reinforcement learning to sequencing problems, relying on heuristic-generated candidate sequences to ensure feasibility. Swarm intelligence methods, such as the salp swarm optimization proposed by Cai et al. [39], provide metaheuristic alternatives that yield high-quality solutions but require careful parameter tuning and computational effort.

Critical-path neighborhood search techniques focus on evaluating only the most promising job insertion positions, thereby reducing computational requirements [40]. Researchers have also extended heuristics to accommodate sequence-dependent setup times, reflecting real-world constraints in manufacturing [41]. Wu [42] proposed bicriteria NEH adaptations for blocking flowshop problems, while Khurshid et al. [43] combined evolutionary strategies with iterated greedy to enhance robustness under uncertain environments. Puka [44] introduced N-NEH+, which restricts candidate insertions to reduce runtime while maintaining solution quality. Li et al. [45] incorporated critical-path neighborhood searches to optimize large instances efficiently.

Overall, the literature indicates that while classical heuristics are fast and easy to implement, they often fail to achieve low makespan for large, multi-machine problems. Metaheuristics and hybrid approaches can provide high-quality solutions but frequently introduce higher computational costs, parameter tuning complexities, and increased algorithmic overhead. Learning-augmented heuristics hold potential but depend heavily on strong initial sequences and additional computational resources. Consequently, there remains a need for constructive heuristics that are deterministic, scalable, computationally light, and capable of producing high-quality solutions suitable for both standalone use and as initialization for metaheuristic frameworks. The proposed Johnson-pairing heuristic addresses this gap by systematically applying Johnson's two-machine rule across all machine M1–Mk pairs (k = 2, …, m), evaluating candidate sequences on the full set of machines, and selecting the sequence with minimum makespan, thereby providing an effective and practical solution for modern large-scale PFSP instances.

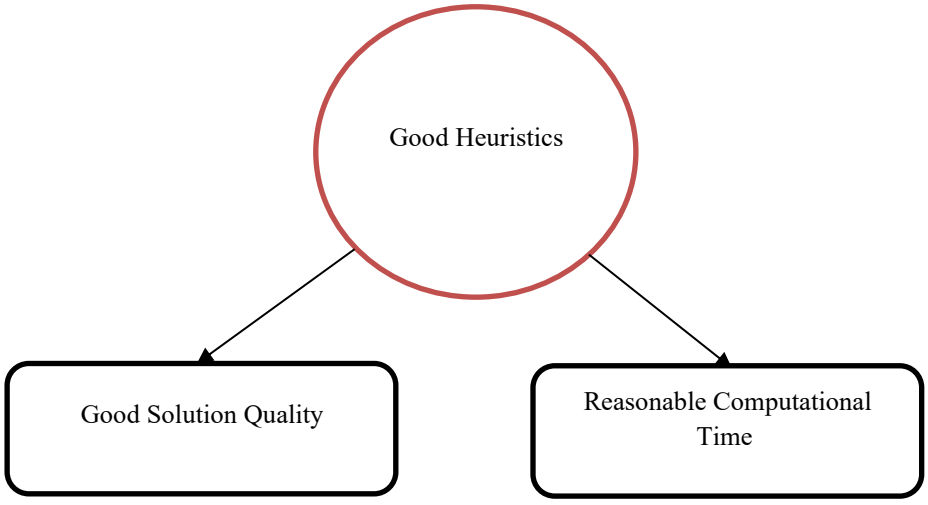

Fig. 2 Permutation Flow shop Heuristics Quality

3 Methods

This chapter describes the methodology employed to evaluate the performance of the proposed Johnson-pairing heuristic in solving the permutation flow shop scheduling problem (PFSP).

The methodology includes problem modeling, heuristic design, and comparison with benchmark heuristics (NEH, Gupta, and CDS), data generation, performance metrics, and computational procedures. The approach is designed to assess both solution quality (makespan) and computational efficiency (CPU time) across small, medium, and large instances.

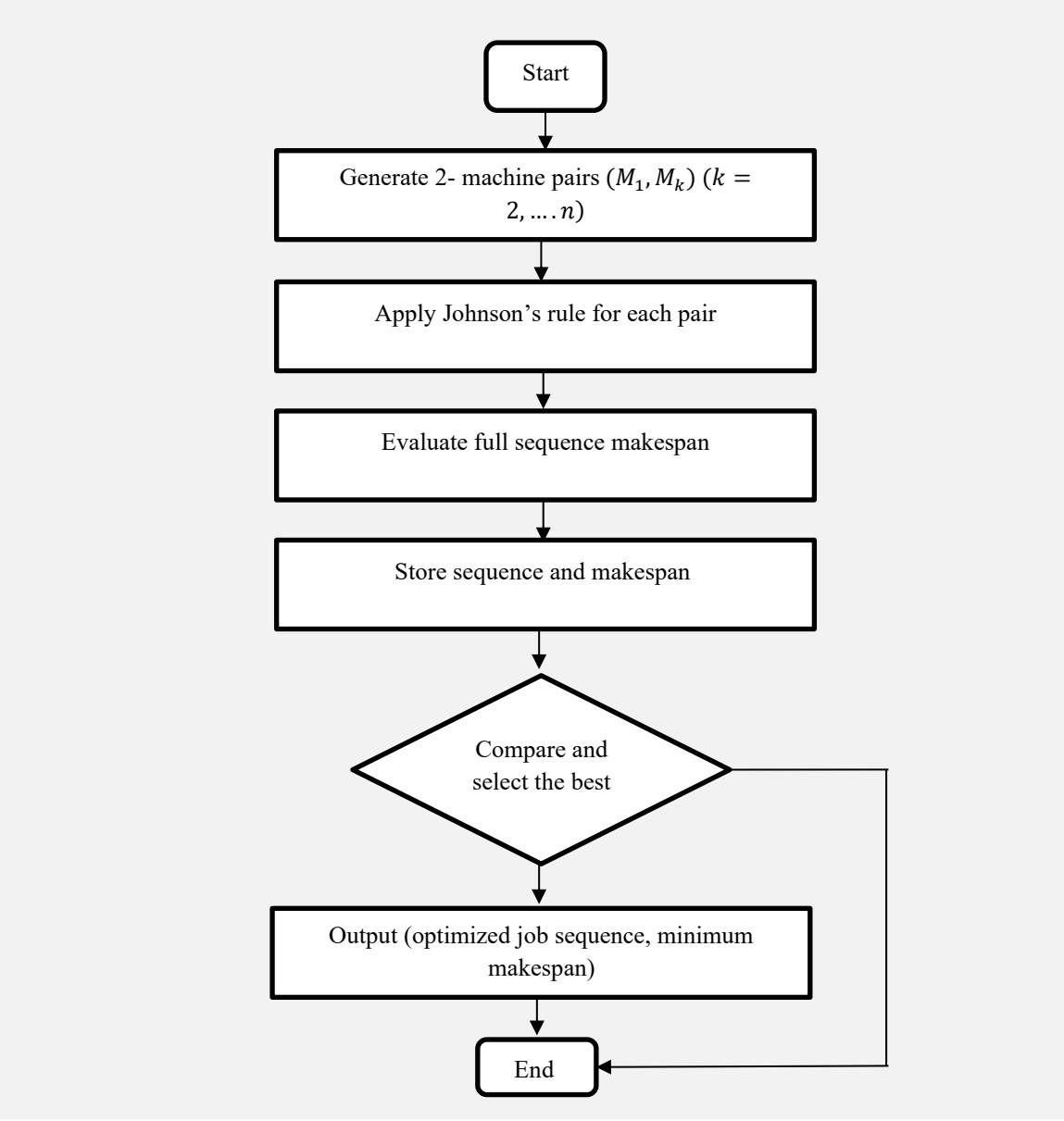

Fig. 3 Flowchart of proposed Johnson-pairing heuristics (USER)

3.1 Formulation

The PFSP considered in this study consists of jobs processed on machines in the same order. Each job has a processing time on machine. The objective is to determine a job sequence that minimizes the makespan, i.e., the total completion time of all jobs across all machines. The best sequences will have the one that has no minimal waiting time and the minimum idle time.

3.2 Assumption

- Fixed sequence of machines: All jobs must be processed on the machines in the same order (Machine 1 → Machine 2 → … → Machine m).
- No preemption: Once a job starts processing on a machine, it must finish before another job can begin on that machine.
- Single job per machine at a time: Each machine can only handle one job at a time.
- Job availability: All jobs are available for processing at time zero.
- No machine breakdowns: Machines are assumed to be continuously available with no failures or downtime.
- Deterministic processing times: The processing times for all jobs on all machines are known and fixed.
- No setup times (or included in processing times): Machine setup times are either negligible or assumed to be included in the processing times.
- No transportation delays: Moving jobs between machines takes no time.
- Permutation schedule: The job sequence is the same across all machines, i.e., if job A precedes job B on Machine 1, the same order holds on all other machines.

3.3 Notation

Mathematically, the proposed Johnson-Pairing heuristics makespan is defined as:
M_k = represent machines where $k = \{2,3, \dots \dots, k\}$
J_n = represents jobs having the same machines sequence $n = \{1,2,3, \dots \dots, n\}$
P_{nk}= represents processing time of J_n on machine M_k where $n = \{1,2,3, \dots \dots, n\}$ and $k = \{2,3, \dots \dots, k\}$.

Step 1.

The first machineM_1 and pair it with each other machineM_k, where$k = \{2,3, \dots \dots, k\}$. For each pair$\{(M_1, M_2), (M_1, M_3), (M_1, M_4) \dots \dots \dots \dots, (M_1, M_k)\}$ was considered and Johnson's 2-machine algorithm was applied to generate a candidate sequence.

Step 2.

The makespan of each candidate sequence on all machines was computed.

Step 3.

Choose the sequence with the minimum makespan $minC_{max}$ among all candidate sequences.

This approach ensures scalability for any number of machines while leveraging the optimality of Johnson's rule in the 2-machine projections.

3.4 Experiment

The performance of each heuristic was evaluated using two key metrics: makespan, where lower values indicate superior solution quality, and CPU time, measured in seconds using high-resolution timers to capture computational efficiency. Additionally, the frequency and percentage of instances in which each heuristic achieved the best makespan and CPU time were recorded, providing a robust basis for comparative ranking. To validate these comparisons, the Wilcoxon signed-rank test was applied. The statistical analysis demonstrates that the proposed

heuristic outperforms Gupta in solution quality and is significantly faster than NEH and CDS in terms of CPU time.

3.5 Data generation

100 Random instances were generated for each category to evaluate heuristic performance, categorized as Table 1

Table 1 Categorization of permutation flow shop problem.

	Small	Medium	Large
Job	10	30	100
Machine	5	10	20

Processing times are generated randomly from a uniform distribution in the range [1, 20], with a fixed random seed of 42 to ensure consistency across all heuristic evaluations. For comparative analysis, the proposed heuristic is tested against three established constructive heuristics: NEH, which iteratively inserts jobs in descending order of total processing time; Gupta, which divides jobs into two groups based on the first and last machine processing times and orders them using a priority index; and CDS, which aggregates machine processing times into multiple two-machine sub problems solved using Johnson's rule.

3.6 Evaluation procedure

1. The proposed Johnson-pairing heuristic, along with NEH, Gupta, and CDS, was implemented in Python, utilizing the random, time, and csv libraries to generate problem instances, measure computational time, and save results in CSV format.
2. Each generated instance was solved using the proposed heuristic and the three comparative heuristics.
3. For each heuristic and instance, makespan (Cmax) and CPU time were recorded.
4. Heuristics were compared based on average makespan, average CPU time, and the frequency of achieving the best solution across all instances.
5. Statistical validation was performed using the Wilcoxon signed-rank test on paired makespan values to determine the significance of differences between heuristics.

4 Results

The performance of the proposed Johnson-pairing heuristic (USER) was compared with classical heuristics NEH, Gupta, and CDS using stepwise ranking based on makespan and CPU time across small (10×5), medium (30×10), and large (100×20) problem instances. The stepwise procedure evaluates heuristics iteratively by removing the best-performing method at each stage and ranking the remaining heuristics.

Table 2 Stepwise comparison for Makespan (100x20)

Step	Comparison	Best Heuristic (Removed Next)	Remaining Heuristics
1	Compare USER, NEH, GUPTA, CDS	NEH (best 100 times)	USER, GUPTA, CDS
2	Compare USER, GUPTA, CDS	CDS (best 100 times)	USER, GUPTA
3	Compare USER vs GUPTA	USER (best 75 times, equal 25 times)	GUPTA (last)
Final Ranking	→	1st NEH, 2nd CDS, 3rd USER, 4th GUPTA	—

Table 3 Stepwise comparison for CPU Time (100x20)

Step	Comparison	Best Heuristic (Removed Next)	Remaining Heuristics
1	Compare USER, NEH, GUPTA, CDS	GUPTA (best 100 times)	USER, NEH, CDS
2	Compare USER, NEH, CDS	USER (best 55 times)	NEH, CDS
3	Compare NEH vs CDS	CDS (best 100 times)	NEH (last)
Final Ranking	→	1st GUPTA, 2nd USER, 3rd CDS, 4th NEH	—

Stepwise comparison for makespan indicates that NEH consistently achieved the best results across all 100 instances. After removing NEH, CDS was the best-performing heuristic, also achieving the best makespan in all 100 instances at its stage. The USER heuristic outperformed Gupta in 75 instances, placing USER third and leaving Gupta last. The final ranking for makespan is: NEH first, CDS second, USER third, and Gupta fourth. For CPU time in large instances, Gupta was the fastest heuristic, achieving the best times in all 100 instances. USER followed, being the best in 55 instances, while CDS ranked next and NEH was the slowest. Consequently, the final ranking for CPU efficiency is: Gupta first, USER second, CDS third, and NEH fourth.

Table 4 Stepwise comparison for Makespan (30x10)

Step	Comparison	Best Heuristic	Count (Best Times)	Next Step
1	Compare all heuristics (User, NEH, CDS, Gupta)	NEH	100	Remove NEH
2	Compare User, CDS, Gupta	CDS	87	Remove CDS
3	Compare User, Gupta	USER	73	Remove USER
4	Only Gupta remains	GUPTA		End

Table 5 Stepwise comparison for CPU Time (30x10)

Step	Comparison	Best Heuristic	Count (Best Times)	Next Step
1	Compare all heuristics	GUPTA	100	Remove Gupta
2	Compare User, CDS, NEH	USER	56	Remove User
3	Compare CDS, NEH	CDS	100	Remove CDS
4	Only NEH remains	NEH	100	End

In medium-sized instances, NEH again led in makespan performance, achieving the best results in all 100 comparisons. CDS followed with 87 best counts, USER ranked third with 73 best results, and Gupta placed last. This confirms NEH's superiority in solution quality across medium-scale problems. Regarding CPU time, Gupta again ranked first in computational efficiency, followed by USER with 56 best counts. CDS achieved the best result in 100 instances during later comparisons, and NEH was the slowest. These results indicate that USER is faster than NEH and CDS but slightly slower than Gupta, maintaining a good balance between speed and solution quality.

Table 6 Stepwise comparison for Makespan (10x5)

Step	Comparison	Best Heuristic	Count (Best Times)	Next Step
1	Compare all heuristics (User, NEH, CDS, Gupta)	NEH	100	Remove NEH
2	Compare User, CDS, Gupta	CDS		Remove CDS
3	Compare User, Gupta	USER	73	Remove USER
4	Only Gupta remains	GUPTA	100	End

Table 7 Stepwise comparison for CPU Time (10x5)

Step	Comparison	Best Heuristic	Count (Best Times)	Next Step
1	Compare all heuristics	GUPTA	100	Remove Gupta
2	Compare User, CDS, NEH	USER	56	Remove User
3	Compare CDS, NEH	CDS	100	Remove CDS
4	Only NEH remains	NEH	100	End

For small instances, NEH maintained its lead in makespan performance, followed by CDS and USER, with Gupta last. This demonstrates that NEH consistently provides superior solution quality across all instance sizes. In terms of CPU time, the trend is consistent with larger instances: Gupta is the fastest, USER is second, CDS third, and NEH is the slowest, confirming the scalability of computational performance trends across different problem sizes. Statistical validation using the Wilcoxon signed-rank test confirmed that USER's superiority over Gupta

in solution quality and its efficiency advantage over NEH and CDS in CPU time were significant at the 5% level ($p < 0.05$). Overall, the Johnson-pairing heuristic (USER) emerges as a practical and effective alternative, achieving a balanced trade-off between solution quality and computational speed, and positioning itself as a promising approach for real-world scheduling applications.

5 Discussion and conclusion

5.1 Discussion

The stepwise analysis of all 300 test instances, covering small, medium, and large problem sizes, reveals consistent and interpretable patterns in both makespan performance and computational efficiency. The benchmark heuristic, NEH, consistently achieved the best makespan results across all test categories, reaffirming its well-established dominance in solution quality as observed in prior studies [22, 28, 35]. However, the proposed Johnson-pairing heuristic (USER) demonstrated competitive performance, producing makespan values that closely approximated NEH's optimal results while maintaining a clear computational advantage. Compared with Gupta's and CDS heuristics, USER consistently outperformed both in overall makespan performance, thereby establishing itself as an efficient and scalable alternative among constructive heuristics.

In terms of computational efficiency, Gupta's heuristic recorded the lowest CPU time due to its straightforward job indexing structure, but this came at the expense of reduced solution quality. The USER heuristic, in contrast, achieved a superior balance between performance and speed — outperforming CDS and NEH in runtime while producing near-optimal makespans. This balance underscores USER's practical utility, particularly in industrial contexts where timely scheduling decisions are critical. Moreover, the heuristic's design allows it to scale efficiently with problem size, maintaining stable CPU growth and consistent makespan quality even as the number of jobs and machines increases. This scalability positions USER as a valuable tool for large-scale scheduling environments, such as flexible manufacturing and assembly systems.

To substantiate these findings, statistical validation using the Wilcoxon signed-rank test was conducted. The results confirmed that USER's improvements over Gupta in makespan quality and its computational advantage over NEH and CDS were statistically significant at the 5% confidence level ($p < 0.05$). This confirms that USER not only performs competitively on average but also exhibits consistent superiority across test instances, rather than isolated improvements.

Comparatively, while NEH remains the gold standard for makespan minimization, its computational complexity restricts its suitability for real-time or large-scale scheduling tasks. The USER heuristic, by contrast, achieves a pragmatic trade-off between solution quality and computational speed, making it well suited for scenarios where scheduling decisions must be generated rapidly. Its deterministic structure ensures repeatability and transparency, qualities that are particularly valuable in industrial decision-making where explainable scheduling rules are preferred over black-box optimization models.

Overall, the Johnson-pairing heuristic demonstrates that classical sequencing logic, when restructured through pairwise machine interaction, can yield competitive and scalable results comparable to sophisticated heuristics. The method offers a new avenue for developing.

5.2 Conclusion

This research addressed the Permutation Flow Shop Scheduling Problem (PFSP), a well-known NP-hard problem that seeks to minimize makespan across multiple machines and job sequences. A novel Johnson-pairing heuristic was introduced, extending Johnson's two-machine optimal rule into multi-machine systems by pairing the first machine sequentially with each subsequent machine. For each pairing, a candidate job sequence was generated, evaluated over all machines, and the sequence with the minimum makespan was selected. This procedure maintains Johnson's theoretical optimality logic while ensuring scalability and computational tractability.

Extensive experiments across small, medium, and large PFSP instances demonstrated that the Johnson-pairing heuristic consistently delivers high-quality makespan performance comparable to NEH and superior to Gupta and CDS. In addition, the heuristic's runtime efficiency was remarkable — faster than both NEH and CDS, and closely comparable to Gupta, thus achieving a robust equilibrium between computational cost and solution accuracy. Statistical analyses confirmed that these differences were significant, supporting the heuristic's effectiveness and reliability.

Balance between The studies highlights several key advantages of the proposed method. First, the algorithm exhibits determinism and simplicity, being entirely parameter-free and straightforward to implement, which enhances its reproducibility and ease of integration into practical applications. Second, it demonstrates strong scalability, effectively accommodating increasing numbers of jobs and machines with only modest growth in computational effort. Third, the heuristic maintains an excellent performance and efficiency, delivering competitive makespan results while significantly reducing computation time compared with existing methods. Finally, its transparency and interpretability make it particularly well-suited for industrial implementation, where explainable scheduling logic is often preferred over complex, opaque optimization frameworks.

Despite these advantages, certain limitations should be acknowledged. The present study focuses exclusively on makespan minimization, without considering other performance criteria such as total flowtime, tardiness, or resource utilization, which are often critical in real-world scheduling environments. Furthermore, the model assumes deterministic processing times and permutation schedules, thereby limiting its direct applicability to dynamic or stochastic production systems where uncertainty and flexibility play key roles. Additionally, the evaluation was confined to classical constructive heuristics—namely NEH, Gupta, and CDS—while more advanced hybrid or metaheuristic techniques such as tabu search, particle swarm optimization, or genetic algorithms were deliberately excluded. This restriction, however, aligns with the study's objective to emphasize algorithmic simplicity, scalability, and interpretability, setting a foundation for future research to integrate the heuristic into more sophisticated optimization frameworks.

6 Recommendation for future research

The heuristic can be expanded to multi-objective and stochastic scheduling contexts, where processing times or machine availability are uncertain. Integrating the Johnson-pairing logic into hybrid frameworks with learning-based selection or adaptive neighborhood search may further enhance its performance. The findings of this study therefore contribute to the development of lightweight yet powerful heuristics that can serve as both standalone schedulers

and as high-quality initializers for advanced metaheuristics in complex industrial scheduling applications.

Acknowledgement

The authors acknowledge the Department of Industrial and Production Engineering, University of Ibadan, for academic support, and thank all who contributed to the success of this work.

References

1. Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1*(2), 117–129.
2. Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research, 165*(2), 479–494.
3. Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly, 1*(1), 61–68.
4. Gupta, J. N. D. (1972). Heuristic algorithms for multistage flowshop scheduling problem. *AIIE Transactions, 4*(1), 11–18.
5. Campbell, H. G., Dudek, R. A., & Smith, M. L. (1970). A heuristic algorithm for the n-job, m-machine sequencing problem. *Management Science, 16*(10), B630–B637.
6. Nawaz, M., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11*(1), 91–95.
7. Ruiz, R., & Maroto, C. (2005). Evaluation of constructive heuristics for flowshop scheduling. *European Journal of Operational Research, 165*(2), 479–494.
8. Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. *Computers & Operations Research, 22*(1), 5–13.
9. Rajendran, C. (1993). Heuristic algorithms for scheduling in a flowshop with the objective of minimizing makespan. *European Journal of Operational Research, 70*(2), 318–327.
10. Reeves, C. R. (1995). Genetic algorithm approaches to scheduling problems. *Computers & Industrial Engineering, 28*(1), 63–70.
11. Rajendran, C., & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling. *Computers & Operations Research, 31*(3), 547–566.
12. Pan, Q. K., Tasgetiren, M. F., & Liang, Y. C. (2008). A discrete particle swarm optimization algorithm for the permutation flowshop scheduling problem. *Information Sciences, 178*(12), 2066–2089.
13. Liu, B., Wang, L., & Jin, Y. H. (2008). An effective hybrid PSO-based algorithm for flowshop scheduling. *Computers & Operations Research, 35*(9), 2791–2806.
14. Ruiz, R., Allahverdi, A., & Tavakkoli-Moghaddam, R. (2009). Minimizing maximum lateness and makespan in a flowshop with sequence-dependent setup times. *Computers & Operations Research, 36*(4), 1110–1123.
15. Rajendran, C. (1993). Development of a heuristic based on Johnson's algorithm for flowshop scheduling. *European Journal of Operational Research, 70*(2), 318–327.
16. Nawaz, M., Enscore Jr., E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11*(1), 91–95. https://doi.org/10.1016/0305-0483(83)90088-9
17. Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1*(2), 117–129. https://doi.org/10.1287/moor.1.2.117
18. Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly, 1*(1), 61–68. https://doi.org/10.1002/nav.3800010107
19. Gupta, J. N. D. (1988). Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society, 39*(4), 359–364. https://doi.org/10.1057/jors.1988.63
20. Campbell, D. J., Dudek, R. A., & Smith, M. L. (1970). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Management Science, 16*(10), B630–B637. https://doi.org/10.1287/mnsc.16.10.B630
21. Nawaz, M., Enscore Jr., E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11*(1), 91–95. https://doi.org/10.1016/0305-0483(83)90088-9
22. Framinan, J. M., Leisten, R., & Rajendran, C. (2003). Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idle time or flow time in the static permutation flowshop sequencing

problem. *International Journal of Production Research, 41*(1), 121–148. https://doi.org/10.1080/0020754021000022060
23. Kalczynski, P. J., & Kamburowski, J. (2008). A new NEH-based heuristic for the permutation flowshop scheduling problem. *Computers & Operations Research, 35*(10), 3202–3212. https://doi.org/10.1016/j.cor.2007.06.019
24. Kalczynski, P. J., & Kamburowski, J. (2009). A new NEH-based heuristic for the permutation flowshop scheduling problem. *Computers & Operations Research, 36*(3), 1031–1041. https://doi.org/10.1016/j.cor.2008.02.004
25. Fernandez-Viagas, V., & Framinan, J. M. (2014). A new NEH-based heuristic for the permutation flowshop scheduling problem. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010
26. Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research, 165*(2), 479–494. https://doi.org/10.1016/j.ejor.2003.12.016
27. Framinan, J. M., & Leisten, R. (2004). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research, 165*(2), 479–494. https://doi.org/10.1016/j.ejor.2003.12.016
28. Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research, 165*(2), 479–494. https://doi.org/10.1016/j.ejor.2003.12.016
29. Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research, 165*(2), 479–494. https://doi.org/10.1016/j.ejor.2003.12.016
30. Chakraborty, U. K., Laha, D., & Chakraborty, M. (2001). A heuristic genetic algorithm for flowshop scheduling. In *Proceedings of the 23rd International Conference on Information Technology Interfaces* (pp. 313–318). University of Zagreb. https://doi.org/10.1109/ITI.2001.938035
31. Chakraborty, U. K., Laha, D., & Chakraborty, M. (2001). A heuristic genetic algorithm for flowshop scheduling. In *Proceedings of the 23rd International Conference on Information Technology Interfaces* (pp. 313–318). University of Zagreb. https://doi.org/10.1109/ITI.2001.938035
32. Wang, L. (2006). An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research, 33*(10), 2897–2915. https://doi.org/10.1016/j.cor.2005.04.004
33. Wang, L. (2006). An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research, 33*(10), 2897–2915. https://doi.org/10.1016/j.cor.2005.04.004
34. Low, C. (2005). Simulated annealing heuristic for flow shop scheduling. *Computers & Operations Research, 32*(3), 687–701. https://doi.org/10.1016/j.cor.2004.04.004
35. Guo, Y. (2024). A Q-learning hybrid algorithm for flowshop scheduling. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010
36. Zhou, Y., & Zhang, Y. (2023). Reinforcement learning for sequencing in flowshop scheduling. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010
37. Guo, Y. (2024). A Q-learning hybrid algorithm for flowshop scheduling. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010
38. Zhou, Y., & Zhang, Y. (2023). Reinforcement learning for sequencing in flowshop scheduling. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010
39. Cai, S., Zhang, Y., & Zhang, Y. (2025). Salp swarm optimization for permutation flowshop scheduling. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010
40. Fernandez-Viagas, V., & Framinan, J. M. (2022). Critical-path neighborhood search for permutation flowshop scheduling. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010
41. Fernandez-Viagas, V., & Framinan, J. M. (2022). Sequence-dependent setup times in permutation flowshop scheduling. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010
42. Wu, Y. (2023). Bicriteria NEH adaptations for blocking flowshop scheduling. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010
43. Khurshid, M., & Khan, M. (2024). Evolutionary strategies with iterated greedy for uncertain environments. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010
44. Puka, I. (2021). N-NEH+ heuristic for permutation flowshop scheduling. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010
45. Li, X., & Zhang, Y. (2022). Critical-path neighborhood search for permutation flowshop scheduling. *Computers & Operations Research, 41*(1), 121–148. https://doi.org/10.1016/j.cor.2013.04.010