International Journal of Applied Operational Research
Vol. 14, No. 3, pp. 61-80, Summer 2026

Journal homepage: ijorlu.liau.ac.ir

# Affinity guided learning iterated greedy for permutation flow shop scheduling: A sugar manufacturing application

A. Olalekan Olasupo*, F. Damilare Ibikunle, A. Jeremiah Amogu

**Received:** 13 May 2026; **Accepted:** 20 June 2026

**Abstract** This study proposes a mutual-love learning iterated greedy (IGML) algorithm for the permutation flow shop scheduling problem (PFSP) with the objective of minimizing makespan. The proposed method integrates a mutual-love learning matrix and affinity-guided destruction and reconstruction to improve the search for promising job sequences. The performance of IGML was evaluated against classical iterated greedy (IG) and the Nawaz–Enscore–Ham (NEH) heuristic using 120 Taillard benchmark instances. IGML achieved the lowest average makespan of 6871.717, compared with 6877.792 for IG and 6899.942 for NEH. Similarly, IGML obtained the lowest average RPD of 3.066%, compared with 3.211% for IG and 3.864% for NEH. The Wilcoxon signed-rank test at the 5% significance level confirmed statistically significant differences between IGML and NEH ($p < 0.001$) and between IGML and IG ($p = 0.0027$). A practical case study involving eight replacement components in a sugar manufacturing workshop was further used to demonstrate the applicability of the proposed approach. The results indicate that IGML provides a statistically significant improvement in PFSP makespan performance over the conventional methods considered.

**Keyword:** Permutation Flow Shop Scheduling, Iterated Greedy, Mutual-Love Learning, Affinity, Makespan, Taillard Benchmark.

## 1 Introduction

Manufacturing organizations operate in increasingly competitive environments where productivity, delivery reliability, and efficient resource utilization determine operational success [1]. To achieve these objectives, production scheduling plays a critical role [2] by determining the sequence in which jobs are processed on available machines.

Effective scheduling contributes to reduced production delays, improved machine utilization, lower operational costs, and enhanced customer satisfaction [3]. Consequently, scheduling has become one of the most extensively studied areas within operations research and industrial engineering.

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

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

**F. Damilare Ibikunle**
Department of Industrial and Production Engineering, University of Ibadan, Nigeria

**A. Jeremiah Amogu**
Department of Industrial and Production Engineering, University of Ibadan, Nigeria

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

Among the various manufacturing scheduling environments, the permutation flow shop scheduling problem (PFSP) has attracted considerable research attention due to its practical relevance and computational complexity [4]. In a PFSP environment, a set of jobs must be processed through a series of machines following the same processing order on every machine [5]. The primary objective is typically to minimize the makespan, defined as the completion time of the last job on the final machine [6]. Applications of PFSP can be found in automotive assembly lines, food processing industries, pharmaceutical production systems, textile manufacturing, metal fabrication workshops, and maintenance engineering operations [7].

The PFSP is recognized as a strongly NP-hard combinatorial optimization problem for three or more machines [8]. As problem size increases, the number of possible job sequences grows factorially, making exact optimization approaches computationally impractical for real-life industrial applications [9].

Researchers have focused on the development of heuristic and metaheuristic approaches capable of producing high-quality solutions within acceptable computational times [10]. One of the most influential constructive heuristics for PFSP is the Nawaz–Enscore–Ham (NEH) heuristic proposed by Nawaz, Enscore, and Ham [11]. The NEH heuristic remains a benchmark algorithm because of its simplicity and ability to generate competitive solutions. Several studies have shown that NEH frequently outperforms other constructive heuristics in makespan minimization problems [12].

NEH follows a greedy construction mechanism and lacks an effective improvement phase, which can limit its ability to escape local optimal and to overcome these limitations, metaheuristic algorithms have gained prominence in PFSP research. Approaches such as genetic algorithms, simulated annealing, tabu search, ant colony optimization, particle swarm optimization, variable neighborhood search, and differential evolution have demonstrated promising performance across various scheduling environments [13].

Among these approaches, the iterated greedy (IG) algorithm proposed by Ruiz and Stützle [14] has emerged as one of the most successful methods for PFSP optimization. The effectiveness of IG stems from its iterative destruction–construction mechanism, which balances exploration and exploitation while maintaining computational efficiency [15].

Subsequent research has produced several enhanced variants of the iterated greedy algorithm. [16] developed adaptive destruction strategies to improve search diversification, while [17] incorporated neighborhood-based improvements to strengthen local search capabilities. Recent studies indicate that IG-based algorithms continue to rank among the most competitive approaches for minimizing makespan in permutation flow shops [18].

A major trend in contemporary optimization research involves integrating learning mechanisms into metaheuristic algorithms. Learning-enhanced metaheuristics utilize information gathered during the search process to guide future decisions and improve convergence behaviour [19], such approaches have demonstrated superior performance in routing, scheduling, and production planning applications because they exploit structural information embedded within problem instances [20].

Recent developments in machine-learning-assisted optimization and adaptive search strategies further highlight the growing importance of intelligent learning mechanisms in combinatorial optimization, within manufacturing environments, job interactions often influence schedule quality. Certain jobs exhibit stronger compatibility when processed consecutively because of similarities in processing requirements, workload distribution, or machine utilization patterns. However, most conventional PFSP algorithms evaluate jobs primarily through objective function values and seldom exploit relational information between

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

jobs during the search process [21]. This creates an opportunity for developing learning-guided optimization approaches capable of capturing and utilizing such relationships.

The practical motivation for this research originates from fabrication activities performed within the maintenance workshop of a sugar manufacturing plant. During maintenance operations, worn or damaged machine components are frequently fabricated or refurbished to support uninterrupted production. In the present study, eight fabrication components are processed sequentially through four workstations comprising cutting, grinding, welding, and drilling operations.

Since all components follow the same processing route, the fabrication process can be represented as a permutation flow shop scheduling problem. Inefficient sequencing of these fabrication jobs increases completion time, delays maintenance activities, and potentially extends equipment downtime and to address this challenge, this study develops an affinity-guided learning iterated greedy algorithm (IGML).

The proposed method extends the classical iterated greedy framework through the incorporation of an affinity-based learning mechanism that captures interaction relationships among jobs and utilizes this information during the destruction–reconstruction process. The objective is to improve makespan performance while preserving the computational efficiency associated with iterated greedy algorithms and the aim of this study is to develop an affinity-guided learning iterated greedy algorithm (IGML) for minimizing makespan in a permutation flow shop scheduling environment applied to sugar manufacturing component fabrication.

Traditional heuristic methods often fails to exploit structural relationships among jobs, while standard metaheuristics rely heavily on stochastic exploration without learning from jobs interactions. Incorporating affinity based learning into the search process provides a mechanism for leveraging hidden dependencies between jobs, thereby improving solution quality and convergence behavior. Performance evaluation considers makespan, Relative Percentage Deviation (RPD), computational time, and win –tie-loss analysis, while stistical significance is assessed using the Wilcoxon Signed Rank Test. The experimental framework is based on 120 Taillard benchmark scheduling instances.

## 2 Literature review

### 2.1 Permutation flow shop scheduling problem

The permutation flow shop scheduling problem (PFSP) remains one of the most extensively studied scheduling problems because of its practical relevance in manufacturing systems, production lines, assembly operations, and maintenance workshops. In PFSP, all jobs follow the same machine sequence, and the objective is commonly the minimization of makespan, total completion time, tardiness, or machine idle time [22].

Due to its NP-hard nature, exact optimization approaches become computationally expensive for medium and large problem instances [23], thereby motivating the development of heuristic and metaheuristic solution methods [24]. Several benchmark studies have demonstrated that makespan minimization significantly influences machine utilization and throughput performance in manufacturing environments. As production systems become increasingly complex, PFSP continues to attract attention as a representative model for real-world scheduling applications.

Constructive heuristics constitute the earliest class of methods developed for PFSP. These methods generate schedules directly without iterative improvement. [23] proposed one of the first optimal procedures for two-machine flow shops, laying the foundation for subsequent

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

scheduling heuristics. Among constructive heuristics, the Campbell–Dudek–Smith (CDS) heuristic represented a significant advancement by extending Johnson's concept to multiple machines [25]. Later studies introduced improved insertion-based procedures aimed at reducing makespan while maintaining computational efficiency. Despite their simplicity, constructive heuristics often struggle to explore the search space adequately and may converge to poor local optima and researchers have increasingly turned to iterative improvement approaches capable of refining initial schedules [26].

## 2.2 Metaheuristic approaches for PFSP

Metaheuristics emerged as powerful alternatives capable of balancing exploration and exploitation. Simulated Annealing was among the earliest metaheuristic approaches successfully applied to PFSP [27]. The method introduced probabilistic acceptance of worse solutions to escape local optima. Genetic Algorithms subsequently became popular due to their population-based search mechanism.

Reeves [28] demonstrated the effectiveness of evolutionary operators for PFSP optimization. Numerous improvements involving hybrid crossover and local search procedures have since been proposed. Tabu search approaches also received considerable attention because of their ability to exploit memory structures during neighborhood exploration [29].

Their effectiveness in escaping cycling behavior contributed significantly to advances in scheduling optimization. Ant colony optimization [30], particle swarm optimization [31], differential evolution [32], and variable neighborhood search [33] further expanded the range of metaheuristic frameworks available for PFSP. Although these approaches often achieve high-quality solutions, many require substantial parameter tuning and computational effort, which limits industrial applicability. The iterated greedy (IG) algorithm introduced by Ruiz and Stützle [14] is widely regarded as one of the most effective PFSP metaheuristics.

The algorithm alternates between destruction and reconstruction phases, where selected jobs are removed and subsequently reinserted using heuristic strategies. The success of IG stems from its simplicity, ease of implementation, and ability to achieve competitive makespan results. Following its introduction, numerous variants have been proposed. Lourenço et al. [26] incorporated local search procedures into the IG framework to enhance intensification. Lin et al. [34] introduced adaptive perturbation mechanisms to improve diversification.

Further study demonstrated that hybrid IG variants outperform many population-based metaheuristics on Taillard benchmark instances and later research has focused on adaptive destruction sizes, self-tuning parameters, and learning-guided reconstruction strategies [35]. These developments suggest that incorporating problem-specific knowledge into the destruction–reconstruction process can significantly improve solution quality.

## 2.3 Learning-based scheduling algorithms

Machine learning and adaptive learning mechanisms have recently gained attention within scheduling research. Rather than relying solely on static heuristics, learning-based methods exploit information generated during the search process to guide future decisions. Zeng et al. [36] proposed reinforcement learning strategies for job sequencing. Tang and Dong [37] integrated deep neural networks with scheduling heuristics to improve decision-making.

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

Similarly, Patel et al. [38] demonstrated that adaptive memory mechanisms can improve convergence performance in complex scheduling environments.

Learning-enhanced optimization has also been successfully applied to flexible job shops, hybrid flow shops, and distributed manufacturing systems [39]. These studies indicate that algorithms capable of extracting and exploiting structural information during optimization often outperform purely stochastic approaches. However, most existing learning-based scheduling methods rely on computationally intensive machine learning models that may be impractical for small and medium manufacturing facilities.

Recent optimization studies have explored the use of relationship information among solution components, an Affinity-based concepts have been investigated in clustering, vehicle routing, production planning, and supply chain optimization [40].

The underlying principle is that certain entities exhibit favorable interactions when grouped together by learning and exploiting these interactions, optimization algorithms can improve search effectiveness in scheduling literature, limited work has explicitly modeled pairwise job affinity within Iterated Greedy frameworks. Most destruction mechanisms remain random or purely cost-based, overlooking potential structural relationships among jobs. Consequently, opportunities remain for developing affinity-guided strategies capable of improving reconstruction quality. The literature reveals three major gaps, first, although Iterated Greedy algorithms have demonstrated strong performance for PFSP, most existing variants employ random or heuristic-based destruction mechanisms without explicitly learning relationships between jobs, second, recent learning-enhanced scheduling methods primarily focus on machine learning or reinforcement learning techniques, which often require extensive computational resources and training datasets [41] and third, little attention has been given to affinity-guided learning mechanisms that dynamically capture interaction strengths among jobs and utilize this information during reconstruction.

Therefore, there is a need for a lightweight learning-enhanced Iterated Greedy framework capable of exploiting job relationships while maintaining the computational efficiency that makes IG attractive for industrial applications.

The proposed affinity-guided learning iterated greedy algorithm (IGML) addresses the identified gap by introducing a mutual-affinity matrix that quantifies interaction strength among jobs. Unlike conventional IG, where job removal and reinsertion are largely random, IGML uses affinity information to guide reconstruction decisions.

## 3 Methods

This study develops and evaluates a scheduling framework for minimizing the makespan of permutation flow shop scheduling problems (PFSPs).

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

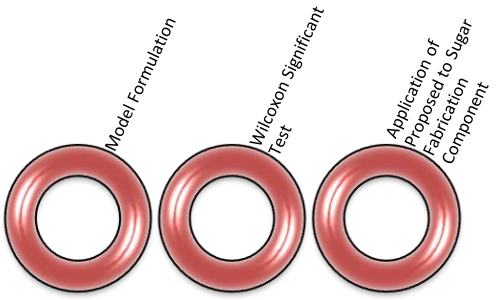


**Fig. 1** Methodology steps

### 3.1 Affinity-guided mutual-love learning iterated greedy algorithm

The assumptions, notations and the formulation of IGML is discussed in this section.

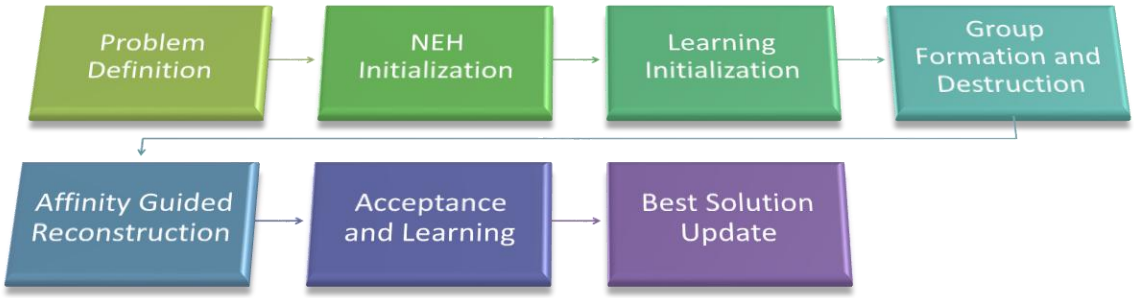


**Fig. 2** IGML formulation procedure

#### 3.1.1 PFSP assumptions

To ensure a deterministic and reproducible scheduling environment, the following assumptions are adopted throughout the PFSP formulation and computational experiments:

- All jobs are available for processing at time zero.
- Each machine can process only one job at a time.
- Every job visits the machines in the same fixed order, from machine 1 through machine m.
- Preemption is not permitted; once processing of a job begins on a machine, it continues until completion.
- Processing times are deterministic and known before scheduling.
- No machine breakdowns are considered.
- Setup times are assumed to be negligible and are therefore not included in the processing-time data.
- Transportation times between machines are assumed negligible.

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

- No maintenance interruption occurs during the scheduling horizon.
- A machine remains idle when the next required job has not completed its operation on the preceding machine.
- The same job permutation is maintained across all machines.
- No parallel processing of a single job is permitted.
- The scheduling objective is exclusively makespan minimization.
- For the proposed stochastic IGML procedure, random operations are controlled using a fixed random seed of 12345 to facilitate reproducibility.

These assumptions correspond to the standard deterministic permutation flow shop environment and ensure that differences in solution quality can be attributed to the scheduling algorithms rather than to stochastic changes in the production environment.

### 3.1.2 Notation

The following notation is used throughout the formulation and implementation of the proposed mutual-love learning iterated greedy (IGML) algorithm.

**Table 1** Notation for proposed IGML

| Symbol | Definition |
|---|---|
| **n** | Number of jobs in the PFSP instance |
| **m** | Number of machines in the PFSP instance |
| **$J_i$** | Job *i*, where $i = 1, 2, ..., n$ |
| **$J_j$** | Job *j*, where $j = 1, 2, ..., n$ |
| **$J_r$** | A removed job selected during reconstruction |
| **$J_s$** | A job remaining in the partial sequence |
| **k** | Machine index, where $k = 1, 2, ..., m$ |
| **$p_{ik}$** | Processing time of job $J_i$ on machine *k* |
| **$P_i$** | Total processing time of job $J_i$ across all machines |
| **$L_{ij}$** | Mutual-learning value between jobs $J_i$ and $J_j$ |
| **$L^0$** | Initial mutual-learning matrix |
| **$A_{ij}$** | Pairwise affinity between jobs $J_i$ and $J_j$ |
| **A(G)** | Average internal affinity of group G |
| **S** | Current job sequence/permutation |
| **$S^0$** | Initial solution generated by NEH |
| **S_NEH** | Final sequence generated by the NEH heuristic |
| **S′** | Partial sequence obtained after destroying the selected group |
| **$S^{cand}$** | Candidate solution obtained after reconstruction |
| **$S^{cur}$** | Current solution during the IGML search |
| **$S^{best}$** | Best solution found during the search |
| **$G_1$, $G_2$** | Two randomly formed job groups |
| **$G^w$** | Weaker group selected for destruction |
| **R** | Set of jobs removed from the current solution |

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

| | |
|---|---|
| $F_r$ | Total affinity of removed job $J_r$ with jobs in the partial sequence |
| **q** | A possible insertion position for a selected job |
| **q*** | Insertion position producing the minimum makespan |
| $C_{max}(S)$ | Makespan of sequence S |
| $C_{max}^{0}$ | Makespan of the initial NEH solution |
| $C_{max}^{cand}$ | Makespan of the candidate solution |
| $C_{max}^{cur}$ | Makespan of the current solution |
| $C_{max}^{best}$ | Makespan of the best solution found so far |
| **Δ** | Difference between candidate and current makespan |
| $P_acc$ | Probability of accepting a worse candidate solution |
| **u** | Random number uniformly generated in [0, 1) |
| **T** | Temperature used in the probabilistic acceptance criterion |
| **A** | Learning increment applied to an accepted adjacent job pair |
| $N_{iter}$ | Maximum number of IGML iterations |
| **R** | Index identifying a removed job during reconstruction |
| **S** | Index identifying a job in the partial sequence |
| **Q** | Position at which a selected job is inserted |
| **RPD** | Relative percentage deviation from the Taillard best-known solution |

### 3.1.3 IGML algorithms formulation

The following steps are involved in Iterated Greedy Mutual Love formulation.

**Step 1: Calculate total job processing time**

Let i = job index, where i = 1, 2, ..., n. k = machine index, where k = 1, 2, ..., m. $p_{ik}$ = processing time of job i on machine k. $P_i$ = total processing time of job i across all machines. The total processing time of each job is calculated by summing its processing times across all machines:

$$P_i = \Sigma_{k=1}^{m} p_{ik} \quad 3.1$$

**Step 2: Construct the initial NEH solution**

The total processing time obtained in Step 1 is used to arrange the jobs in descending order. Let $P_i$ = total processing time of job i. L = ordered list of jobs according to descending $P_i$. $S^0$ = initial solution generated by NEH. S_NEH = final sequence obtained by NEH. The jobs are first sorted according to $P_{i1} \geq P_{i2} \geq ... \geq P_{in}$ Therefore, the ordered job list is represented as $L = (i_1, i_2, ..., i_n)$ where $i_1$ is the job with the largest total processing time and $i_n$ is the job with the smallest total processing time.

NEH starts with the first job $S_1 = (i_1)$. The next job is then inserted into every possible position. For the r-th job, let $S_r(q)$ = candidate sequence obtained by inserting job $i_r$ at position q in the current sequence. The makespan of every candidate sequence is calculated, and the position producing the smallest makespan is selected:

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

$$q = \arg\min \mathrm{Cmax}(S_r(q)) * \quad 3.2$$

The selected sequence becomes the new current sequence $S_r = S_r(q)*$. This insertion process is repeated until all n jobs have been inserted. Consequently, the final NEH sequence is $S^0$ = S_NEH and its corresponding makespan is $Cmax^0$ = Cmax(S_NEH) The resulting NEH solution is used as the initial solution for IGML S_current = $S^0$ and S_best = $S^0$

**Step 3: Initialize the learning matrix**

After obtaining the initial NEH solution, the mutual-love learning matrix is initialized to represent the initial relationship between every pair of different jobs. Let $L_{ij}$ = learning value between job i and job j. i, j = job indices, where i, j = 1, 2, ..., n. The initial learning matrix is defined as $L_{ij} = 1$, if $i \neq j$ and $L_{ii} = 0$, if i = j. Therefore, the complete initialization rule is $L_{ij}$ = {0, if i = j; 1, if $i \neq j$} for n jobs, the initial learning matrix is therefore an n × n matrix:

$L^0$ =

| | $J_1$ | $J_2$ | ... | $J_n$ |
|---|---|---|---|---|
| $J_1$ | 0 | 1 | ... | 1 |
| $J_2$ | 1 | 0 | ... | 1 |
| ⋮ | ⋮ | ⋮ | ⋱ | ⋮ |
| $J_n$ | 1 | 1 | ... | 0 |

The diagonal elements are set to zero because a job is not compared with itself. All off-diagonal elements are initially set to one because no pair of jobs has yet received any preferential learning. $L^0$ = initial learning matrix. This matrix is subsequently updated during the IGML search whenever job pairs occur consecutively in an accepted solution.

**Step 4: Calculate pairwise affinity**

After initializing the learning matrix, the pairwise affinity between every two different jobs is calculated by combining their learning value with the difference in their total processing times. Let $A_{ij}$ = affinity between job i and job j. $L_{ij}$ = learning value between job i and job j. $P_i$ = total processing time of job i. $P_j$ = total processing time of job j. $|P_i - P_j|$ = absolute difference between the total processing times of jobs i and j. The affinity is calculated as:

$$A_{ij} = L_{ij} / [1 + |P_i - P_j|] \quad 3.3$$

The term $1 + |P_i - P_j|$ prevents division by zero when two jobs have identical total processing times. The calculation is performed for every pair of different jobs $A_{ij} = L_{ij} / [1 + |P_i - P_j|]$, $i \neq j$ for the same job: $A_{ii} = 0$. Therefore, the initial affinity matrix is: $A^0$ =

| | $J_1$ | $J_2$ | ... | $J_n$ |
|---|---|---|---|---|
| $J_1$ | 0 | $A_{12}$ | ... | $A_{1n}$ |
| $J_2$ | $A_{21}$ | 0 | ... | $A_{2n}$ |
| ⋮ | ⋮ | ⋮ | ⋱ | ⋮ |
| $J_n$ | $A_{n1}$ | $A_{n2}$ | ... | 0 |

Since the initial learning matrix is symmetric $L_{ij} = L_{ji}$ the resulting affinity matrix is also symmetric $A_{ij} = A_{ji}$. Thus, a pair of jobs has higher affinity when its learning value is high and the difference between their total processing times is small. This affinity matrix is subsequently used to identify the weaker group during the destruction phase and to select removed jobs during reconstruction.

**Step 5: Randomly divide the current solution into two groups**

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

Let the current solution be $S = (J_1, J_2, \ldots, J_n)$. The jobs in S are randomly rearranged and divided into two groups: $S \rightarrow G_1$ and $G_2$ where $G_1$ and $G_2$ are the two randomly formed job groups.

**Step 6: Calculate the affinity of each group**

The affinity of a group is determined from the pairwise affinities of the jobs contained in that group. For a group G containing at least two jobs, its average internal affinity is

$$A(G) = [\Sigma A_{ij}] / [number\ of\ distinct\ job\ pairs\ in\ G] \quad 3.4$$

Where $A_{ij}$ is the affinity between jobs $J_i$ and $J_j$. For the implemented group size of two, if $G = \{J_i, J_j\}$, then the group affinity is simply $A(G) = A_{ij}$. Thus, calculate $A(G_1)$ and $A(G_2)$.

**Step 7: Select the weaker group**

The group with the smaller internal affinity is considered the weaker group. Therefore,

$$G^w = arg\ min\ \{A(G_1), A(G_2)\} \quad 3.5$$

In words, $G^w$ is the group having the minimum affinity. If $A(G_1) < A(G_2)$, then $G^w = G_1$. If $A(G_2) < A(G_1)$, then $G^w = G_2$. If the two affinities are equal, the implementation randomly selects one of the two groups.

**Step 8: Destroy the weaker group**

The jobs belonging to the weaker group $G^w$ are removed from the current solution S. The resulting partial sequence is $S' = S - G^w$. The removed jobs are stored in a removed-job set: $R = G^w$. The jobs in R are then randomly rearranged before reconstruction. Thus, after destruction $S = S' \cup R$.

**Step 9: Calculate the affinity of each removed job with the partial sequence**

For every removed job $J_r \in R$, its affinity with all jobs remaining in S′ is calculated. The total affinity of removed job $J_r$ is

$$F_r = \Sigma A_{rs}, for\ J_s \in S'. \quad 3.6$$

Here $J_r$ is the removed job being evaluated. $J_s$ is each job currently remaining in the partial sequence, $A_{rs}$ is the affinity between $J_r$ and $J_s$. $F_r$ is the total affinity of $J_r$ with the partial sequence. Therefore, the affinity calculation is repeated for every removed job.

**Step 10: Select the strongest removed job**

The removed job having the largest total affinity with the current partial sequence is selected first for reconstruction. Therefore $J_r = arg\ max\ F_r, for\ J_r \in R.*$. In words, $J_r*$ is the removed job that has the strongest overall affinity with the jobs already present in S′. If two or more removed jobs have the same maximum affinity, the implementation randomly selects one of them.

**Step 11: Test every possible insertion position**

After selecting $J_r*$, the algorithm tests every possible position in the partial sequence S′. For each possible position q, the selected job is inserted into S′ to form a candidate sequence $S'q = insert(J_r, q).*$ The makespan of every candidate sequence is then calculated. The best insertion position is $q = arg\ min\ Cmax(S'q).*$ Thus, q* is the position that produces the smallest makespan after inserting the selected job.

**Step 12: Insert the selected job**

The selected job $J_r*$ is permanently inserted at the best position q*. Therefore, the partial sequence is updated as $S' \leftarrow insert(J_r, q).**$ The selected job is then removed from the set of jobs awaiting reconstruction $R \leftarrow R - \{J_r\}.*$

**Step 13: Repeat the affinity-guided reconstruction**

If removed jobs still remain, $R \neq \emptyset$, the algorithm returns to Step 9. For each remaining removed job, its affinity with the newly enlarged partial sequence is recalculated. The job with the highest affinity is selected, all possible insertion positions are tested, and

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

the position producing the minimum makespan is chosen. This process continues until $R = \emptyset$. At this point, all destroyed jobs have been reconstructed and the resulting sequence is the candidate solution: $S^{cand} = S'$

**Step 14: Evaluate the candidate solution**

After all removed jobs have been reconstructed, the resulting sequence is the candidate solution, denoted by $S^{cand} = S'$. The makespan of the candidate solution is then calculated as $C_{max}^{cand} = C_{max}(S^{cand})$. Here, $C_{max}^{cand}$ represents the completion time of the last job on the last machine for the reconstructed candidate sequence.

**Step 15: Apply the acceptance criterion**

The candidate solution is compared with the current solution. Let $C_{max}^{cur}$ denote the makespan of the current solution. If $C_{max}^{cand} \leq C_{max}^{cur}$, the candidate solution is accepted directly because it is equal to or better than the current solution. Thus, $S^{cur} \leftarrow S^{cand}$ and $C_{max}^{cur} \leftarrow C_{max}^{cand}$. If the candidate solution is worse, such that $C_{max}^{cand} > C_{max}^{cur}$, the difference in makespan is calculated as

$$\Delta = C_{\max}^{\text{cand}} - C_{\max}^{\text{cur}} \quad 3.7$$

The probability of accepting the worse candidate is then calculated as

$$P_{\text{a}}cc = exp(-\Delta/T). \quad 3.8$$

Here, T is the temperature parameter used to control the probability of accepting a worse solution. A random number u is generated from the interval $0 \leq u < 1$. The candidate solution is accepted when $u < P_a cc$. If $u \geq P_a cc$, the candidate solution is rejected and the current solution remains unchanged. Therefore, the acceptance mechanism allows the algorithm to occasionally accept a worse solution and thereby continue exploring different regions of the solution space.

**Step 16: Update the mutual-love learning matrix**

When the candidate solution is accepted, the accepted sequence becomes the new current solution. The learning matrix is then updated using every pair of adjacent jobs in the accepted sequence. Suppose two consecutive jobs in the accepted sequence are $J_i$ and $J_j$. Their learning values are updated as $L_{ij} \leftarrow L_{ij} + \alpha$ and $L_{ji} \leftarrow L_{ji} + \alpha$. Here, $\alpha$ is the learning increment, which is set to 0.10 in the implementation. For example, if the accepted sequence contains the adjacent relationship $J_2 \rightarrow J_5$, then $L_{25} \leftarrow L_{25} + \alpha$ and $L_{52} \leftarrow L_{52} + \alpha$. The same procedure is applied to every adjacent pair in the accepted sequence. Thus, if $S^{cur} = (J_1, J_4, J_2, J_5)$, the adjacent relationships are $(J_1,J_4)$, $(J_4,J_2)$, and $(J_2,J_5)$. The corresponding learning values are therefore updated as $L_{14} \leftarrow L_{14} + \alpha$, $L_{41} \leftarrow L_{41} + \alpha$, $L_{42} \leftarrow L_{42} + \alpha$, $L_{24} \leftarrow L_{24} + \alpha$, $L_{25} \leftarrow L_{25} + \alpha$, $L_{52} \leftarrow L_{52} + \alpha$. This update strengthens relationships that repeatedly occur as adjacent jobs in accepted solutions. Importantly, the learning update is performed whenever a solution is accepted, whether the accepted solution is better or worse than the previous current solution.

**Step 17: Preserve the best solution**

The algorithm maintains a separate best-so-far solution so that acceptance of a worse solution cannot cause the best solution previously discovered to be lost. Let $S^{best}$ represent the best solution found so far and let $C_{max}^{best}$ represent its makespan. After the acceptance decision, the current solution is compared with the best solution. If $C_{max}^{cur} < C_{max}^{best}$, the current solution becomes the new best solution: $S^{best} \leftarrow S^{cur}$ and $C_{max}^{best} \leftarrow C_{max}^{cur}$. If $C_{max}^{cur} \geq C_{max}^{best}$, the best solution remains unchanged. Therefore, the algorithm always retains the lowest makespan obtained during the search.

**Step 18: Repeat the IGML search process**

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

After the acceptance, learning, and best-solution update, one IGML iteration is completed. The algorithm then returns to the destruction stage and repeats the following sequence: current solution → two-group formation → weak-group identification → destruction → affinity-guided reconstruction → candidate evaluation → acceptance → learning → best-solution update. The complete search process is repeated for $N_{iter}$ = 50 Iterations. At the end of the 50 iterations, the algorithm terminates and returns $S^{best}$ as the final IGML scheduling sequence and $C_{max}^{best}$ as its corresponding makespan. Thus, the final output of IGML is the best permutation discovered during the 50-iteration search, rather than necessarily the solution accepted in the final iteration.50 Iteration was the same condition set for Iterated Greedy.

### 3.2 Statistical significance test

The Wilcoxon signed-rank test was used to determine whether the differences in makespan between the algorithms were statistically significant. The test was selected because the same PFSP benchmark instances were evaluated by each algorithm, producing paired observations, and because the test does not require the paired differences to follow a normal distribution. Let t = 1, 2, ..., N denote the benchmark instance, where N = 120, and let $C_t^A$ and $C_t^B$ denote the makespans obtained by two compared algorithms A and B for instance t. The paired difference is calculated as

$$D_t = C_t^A - C_t \quad 3.9$$

Where a positive value indicates that algorithm B obtained a lower makespan. The zero differences are excluded, and the absolute values of the remaining differences are ranked:

$$R_t = rank(|D_t|) \quad 3.10$$

The positive and negative rank sums are calculated as

$$W^+ = \Sigma R_t, D_t > 0 \quad 3.11$$

$$W^- = \Sigma R_t, D_t < 0 \quad 3.12$$

and the Wilcoxon statistic is W = min($W^+$, $W^-$). The following hypotheses were tested $H_0$: median($D_t$) = 0**.** $H_1$: median($D_t$) ≠ 0 A significance level of α = 0.05 was adopted. Therefore**,** p < 0.05 → reject $H_0$**.** p ≥ 0.05 → fail to reject $H_0$. Three paired comparisons were performed NEH versus IGML, IG versus IGML, and NEH versus IG. Since makespan is minimized, the number of wins, ties, and losses was also determined from the paired differences. The percentage improvement of algorithm B over algorithm A was calculated as

$$Improvement\ (\%) = [(C^A - C^B) / C^A] \times 100 \quad 3.13$$

Where $\bar{C}^A$ and $\bar{C}^B$ are the mean makespans of algorithms A and B, respectively. This procedure provides both the magnitude of the observed performance difference and evidence of whether the difference is statistically significant across the benchmark instances.

### 3.3 Application of the proposed IGML algorithm to the sugar manufacturing case study

The proposed IGML algorithm was further evaluated using a real-world-inspired scheduling case from a sugar factory maintenance workshop. The workshop fabricates eight replacement

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

components, represented as jobs $J_1$–$J_8$, using four workstations: cutting ($M_1$), grinding ($M_2$), welding ($M_3$), and drilling ($M_4$). All jobs follow the identical routing sequence.

$$M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow M_4$$

And therefore constitute a four-machine permutation flow shop scheduling problem.

### 3.3.1 Case study data

The eight fabrication components considered in the workshop are shown in Table 2.

**Table 2** Fabrication components

| Job | Component description |
|---|---|
| $J_1$ | Conveyor Shaft Bracket |
| $J_2$ | Roller Support Frame |
| $J_3$ | Cane Carrier Guide Plate |
| $J_4$ | Gearbox Mounting Plate |
| $J_5$ | Pump Coupling Housing |
| $J_6$ | Bearing Support Block |
| $J_7$ | Conveyor Chain Tension Arm |
| $J_8$ | Mill Scraper Holder |

The processing time required by each job on each workstation was recorded in minutes and organized into the processing-time matrix shown in Table 3.

**Table 3** Processing Time Matrix (minutes)

| $p_{ik}$ Job | $M_1$: Cutting | $M_2$: Grinding | $M_3$: Welding | $M_4$: Drilling |
|---|---|---|---|---|
| $J_1$ | 12 | 18 | 25 | 10 |
| $J_2$ | 15 | 22 | 30 | 12 |
| $J_3$ | 10 | 15 | 20 | 8 |
| $J_4$ | 18 | 25 | 35 | 15 |
| $J_5$ | 14 | 20 | 28 | 11 |
| $J_6$ | 11 | 17 | 24 | 9 |
| $J_7$ | 16 | 23 | 32 | 13 |
| $J_8$ | 13 | 19 | 27 | 10 |

Accordingly, the processing time of job $J_i$ on machine $M_k$ is denoted by, where i = 1, ..., 8 and k = 1, ..., 4.

### 3.3.2 Scheduling model and application of NEH, IG and IGML

The sugar manufacturing case study was formulated as a permutation flow shop scheduling problem involving eight jobs ($J_1$–$J_8$) and four machines ($M_1$–$M_4$), with all jobs following the routing sequence $M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow M_4$ and using the processing times presented in Table 3.

The objective was to determine the permutation of the eight components that minimizes the makespan, with the completion-time and makespan formulations following the PFSP model defined previously.

The same processing-time matrix was supplied to NEH, classical IG, and the proposed IGML to ensure a consistent comparison. NEH was first applied by ordering the jobs according

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

to their total processing times and sequentially inserting them at the positions producing the minimum makespan, with the resulting NEH sequence serving as the initial solution for both IG and IGML.

Classical IG subsequently applied its standard destruction and NEH-based reconstruction procedure, whereas IGML employed the proposed mutual-love learning matrix and affinity-guided destruction and reconstruction mechanisms described in Sections 3.1. For the case study, the same IGML parameter settings used in the benchmark experiments were retained, including a learning increment of $\alpha = 0.10$ and 50 search iterations, thereby maintaining consistent experimental conditions.

The final job sequence, machine completion times, and makespan produced by each algorithm were recorded and compared, with the algorithm producing the lowest makespan regarded as the best-performing approach for the sugar manufacturing case

## 4 Results

### 4.1 Benchmark performance

The performance of NEH, classical IG, and the proposed IGML was evaluated using 120 Taillard permutation flow shop benchmark instances. The results show a progressive improvement in solution quality from NEH to IG and subsequently to IGML. NEH obtained an average makespan of 6899.942, an average RPD of 3.864%**,** and an average CPU time of 18.402 **s**. Classical IG improved the average makespan to 6877.792 and the average RPD to 3.211%, with an average CPU time of 35.598 s.

The proposed IGML achieved the lowest average makespan of 6871.717 and the lowest average RPD of 3.066%. However, this improvement required an average CPU time of 244.832 s. Overall, IGML reduced the average makespan by 0.41% relative to NEH and by 0.09% relative to classical IG. The corresponding RPD reductions were 20.65% relative to NEH and 4.50% relative to IG.

**Table 4** NEH, IG, IGML comparison

| Algorithm | Average Makespan | Average RPD (%) | Average CPU time (s) |
|---|---|---|---|
| NEH | 6899.942 | 3.864 | 18.402 |
| IG | 6877.792 | 3.211 | 35.598 |
| IGML | 6871.717 | 3.066 | 244.832 |

### 4.2 Statistical significance

The Wilcoxon signed-rank test was used to determine whether the observed differences in makespan were statistically significant across the 120 matched benchmark instances. At $\alpha = 0.05$, all three pairwise comparisons produced statistically significant results. For NEH versus IGML, IGML recorded 94 wins, 26 ties, and no losses, with a p-value of $3.79 \times 10^{-17}$. For IG versus IGML, IGML recorded 54 wins, 40 ties, and 26 losses, with a p-value of 0.00274. For NEH versus IG, IG recorded 88 wins, 32 ties, and no losses, with a p-value of $3.71 \times 10^{-16}$. Since all p-values are below 0.05, the null hypothesis was rejected in each comparison.

These results confirm that the differences in makespan between the three algorithms are statistically significant

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

**Table 5** Wilcoxon significant

| Comparison | Reference Algorithm | Compared Algorithm | Compared Algorithm Wins | Ties | Compared Algorithm Losses | Improvement (%) | Wilcoxon Statistic | p-value | Decision |
|---|---|---|---|---|---|---|---|---|---|
| NEH vs IGML | NEH | IGML | 94 | 26 | 0 | 0.409 | 0 | $3.79 \times 10^{-17}$ | Significant |
| IG vs IGML | IG | IGML | 54 | 40 | 26 | 0.088 | 995.5 | 0.00274 | Significant |
| NEH vs IG | NEH | IG | 88 | 32 | 0 | 0.321 | 0 | $3.71 \times 10^{-16}$ | Significant |

## 4.3 Application to sugar fabrication component

Proposed heuristics and benchmark were applied to sugar fabrication data and Figure 3, 4, 5 indicate the Gantt chart of the schedule

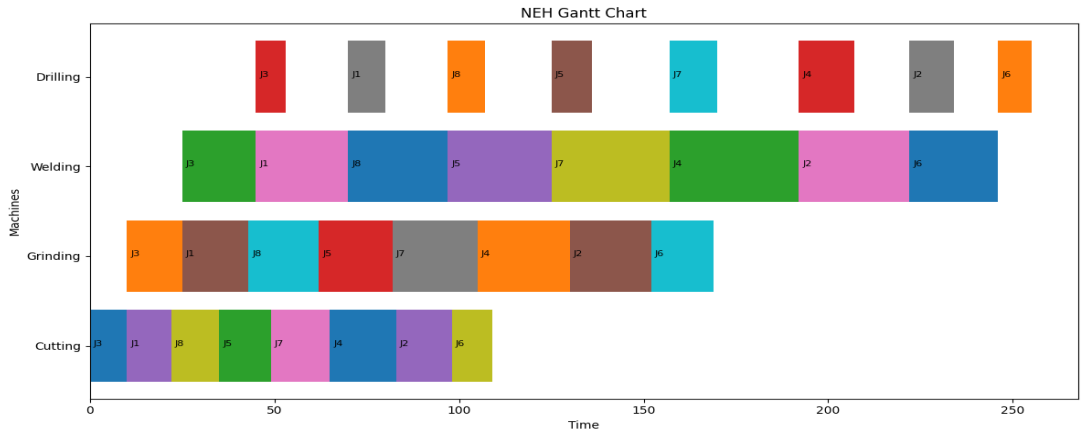


**Fig. 3** NEH Gantt chart

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

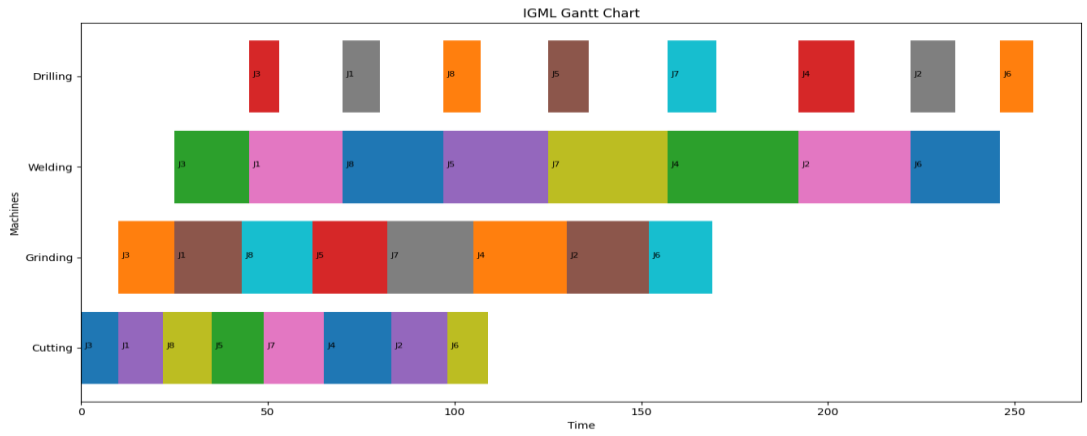


**Fig. 4** IG Gantt chart

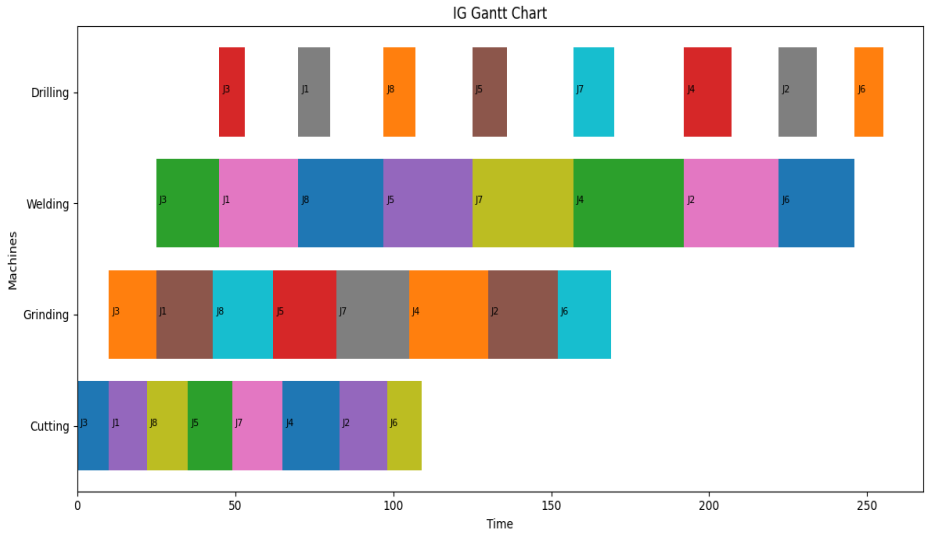


**Fig. 5** IGML Gantt chart

The computational results revealed that the NEH, IG, and IGML algorithms achieved an identical makespan of 255 minutes for the permutation flow shop scheduling instance derived from the sugar manufacturing fabrication workshop. This indicates that all three methods converged to the same scheduling solution for the considered industrial case study.

## 5 Discussion and conclusion

### 5.1 Discussion

The findings indicate that incorporating learning and affinity information into the Iterated Greedy framework provides a useful mechanism for guiding the search process. The improvement of IGML over classical IG suggests that the proposed method does more than simply perform additional iterations of destruction and reconstruction.

The learning matrix preserves information about job relationships observed in previously accepted schedules, while the affinity mechanism uses this information to influence which jobs are disrupted and how they are subsequently reconstructed. This provides a form of search

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

memory that is absent from the conventional IG procedure. The destruction mechanism is important because conventional random destruction may remove jobs without considering their relationships with other jobs. In IGML, the current solution is divided into two groups and the group with weaker internal affinity is selected for destruction. This directs the perturbation toward job relationships that have received comparatively less support during the search.

The subsequent affinity-guided reconstruction further exploits the information accumulated in the learning matrix by prioritizing removed jobs according to their relationship with the jobs already present in the partial sequence. Consequently, the proposed method balances diversification through destruction and intensification through learned affinity.

The statistical analysis is also important in interpreting the observed improvement. Although the numerical difference between IGML and classical IG is relatively small, the paired statistical test indicates that the difference is unlikely to have occurred randomly across the benchmark instances. This suggests that the proposed learning mechanism contributes measurable information to the search rather than merely producing an improvement attributable to experimental variation. The computational results reveal a limitation of the proposed approach.

The additional learning, affinity calculations, group evaluation, and repeated insertion evaluations introduce considerable computational overhead. Therefore, the advantage of IGML is primarily associated with solution quality rather than computational speed. In applications where a scheduling decision must be generated almost immediately, classical IG or NEH may remain preferable. Conversely, where better schedules justify additional computational effort, the proposed approach becomes more attractive.

The practical case study also demonstrates that the methodology is not restricted to abstract benchmark instances. The same PFSP formulation can be applied to the sequencing of replacement components in a manufacturing workshop. The case study therefore illustrates how the algorithm can translate the computational search mechanism into a production scheduling decision. However, the simplified case-study assumptions mean that factors such as machine breakdowns, setup times, maintenance interruptions, and material availability are not represented.

These factors would need to be incorporated before applying the model directly to a more complex industrial environment. The findings support the use of learned job relationships as an additional source of information in Iterated Greedy scheduling. The main contribution is therefore not simply a lower makespan, but the introduction of a mechanism through which information accumulated during the search is reused to guide subsequent destruction and reconstruction decisions

## 5.2 Conclusion

This study proposed an Affinity-Guided Mutual-Love Learning Iterated Greedy algorithm for permutation flow shop scheduling. The method integrates a learning matrix with an affinity measure to guide the destruction and reconstruction stages of Iterated Greedy.

The experimental investigation demonstrates that the proposed approach can improve solution quality over both the constructive NEH method and classical Iterated Greedy. The statistical analysis further establishes that the observed pairwise differences are significant at the adopted significance level.

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

The sugar manufacturing application additionally demonstrates the applicability of the framework to a practical component-sequencing problem. The main limitation is the additional computational effort introduced by the learning and affinity-guided procedures.

Thus, IGML is most appropriate for scheduling environments where obtaining a high-quality production sequence is more important than minimizing computational time.

## 6 Recommendations

Future research should focus on improving the computational efficiency of IGML while retaining its learning capability. In particular, adaptive control of the number of iterations and destruction size could reduce unnecessary computations. Further studies should also investigate alternative affinity formulations and learning-update strategies to determine whether stronger improvements can be obtained.

## Acknowledgement

The authors acknowledge that no external funding or financial support was received for the conduct of this research. The study was completed independently using the resources available to the researchers

## References

1. Sarmiento, R., Byrne, M., Rene Contreras, L., & Rich, N. (2007). Delivery reliability, manufacturing capabilities and new models of manufacturing efficiency. Journal of Manufacturing Technology Management, 18(4), 367-386. https://doi.org/10.1108/17410380710743761
2. Ma, H., Huang, X., Hu, Z., Chen, Y., Qian, D., Deng, J., & Hua, L. (2023). Multi-objective production scheduling optimization and management control system of complex aerospace components: a review. The International Journal of Advanced Manufacturing Technology, 127(11), 4973-4993. https://doi.org/10.1007/s00170-023-11707-4
3. John, B. I. (2020). Integration of intelligent scheduling optimization systems improving production flow, minimizing delays, and maximizing throughput across large-scale industrial operations. Global Journal of Engineering and Technology Advances, 5(3), 156-169. https://doi.org/10.30574/gjeta.2020.5.3.0118
4. Rossit, D. A., Tohmé, F., & Frutos, M. (2018). The non-permutation flow-shop scheduling problem: a literature review. Omega, 77, 143-153. https://doi.org/10.1016/j.omega.2017.05.010
5. Cheng, L., Wang, L., Cai, J., Hu, K., Xiong, Y., & Xia, Q. (2026). Collaborative assembly factory in the distributed permutation flow-shop scheduling problem considering flexible assembly and fuzzy processing time. Engineering Optimization, 58(2), 362-393. https://doi.org/10.1080/0305215x.2025.2466827
6. Olalekan Olasupo, A., Olasunkanmi, E., & Ogunfuye, O. (2025). A fast and scalable heuristic for makespan minimization in permutation flowshop scheduling. International Journal of Applied Operational Research-An Open Access Journal, 13(3), 61-74. https://doi.org/10.71885/ijorlu-2025-3-708
7. Fernandez-Viagas, V., & Framinan, J. M. (2019). A best-of-breed iterated greedy for the permutation flowshop scheduling problem with makespan objective. Computers & Operations Research, 112, 104767. https://doi.org/10.1016/j.cor.2019.104767
8. 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
9. Georgiadis, G. P., Elekidis, A. P., & Georgiadis, M. C. (2019). Optimization-based scheduling for the process industries: From theory to real-life industrial applications. Processes, 7(7), 438. https://doi.org/10.3390/pr7070438
10. Hussain, K., Mohd Salleh, M. N., Cheng, S., & Shi, Y. (2019). Metaheuristic research: a comprehensive survey. Artificial intelligence review, 52(4), 2191-2233. https://doi.org/10.1007/s10462-017-9605-z

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

11. 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
12. de Athayde Prata, B., Nagano, M. S., Martarelli Fróes, N. J., & de Abreu, L. R. (2023, December). The seeds of the neh algorithm: an overview using bibliometric analysis. In Operations Research Forum (Vol. 4, No. 4, p. 98). Cham: Springer International Publishing. https://doi.org/10.1007/s43069-023-00276-7
13. Youssef, H., Sait, S. M., & Adiche, H. (2001). Evolutionary algorithms, simulated annealing and tabu search: a comparative study. Engineering Applications of Artificial Intelligence, 14(2), 167-181. https://doi.org/10.1016/s0952-1976(00)00065-8
14. Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European journal of operational research, 177(3), 2033-2049.
15. Qin, H., Han, Y., Chen, Q., Wang, L., Wang, Y., Li, J., & Liu, Y. (2023). Energy-efficient iterative greedy algorithm for the distributed hybrid flow shop scheduling with blocking constraints. IEEE Transactions on Emerging Topics in Computational Intelligence, 7(5), 1442-1457. https://doi.org/10.1109/tetci.2023.3271331
16. Zeng, L., Ding, Z., Shi, J., & Wang, S. (2024). A discrete multi-objective squirrel search algorithm for energy-efficient distributed heterogeneous permutation flowshop with variable processing speed. Computers, Materials and Continua, 81(1), 1757-1787. https://doi.org/10.32604/cmc.2024.055574
17. Prata, B. D. A., & Nagano, M. S. (2023). A novel iterated greedy algorithm for no-wait permutation flowshop scheduling to minimize weighted quadratic tardiness. Engineering Optimization, 55(12), 2070-2083. https://doi.org/10.1080/0305215x.2022.2144274
18. Framinan, J. M., Perez-Gonzalez, P., & Fernandez-Viagas, V. (2024). Assessing the level of centralisation in scheduling decisions: The role of hybrid approaches. Journal of Industrial Information Integration, 41, 100682. https://doi.org/10.1016/j.jii.2024.100682
19. Talbi, E. G. (2021). Machine learning into metaheuristics: A survey and taxonomy. ACM computing surveys (CSUR), 54(6), 1-32. https://doi.org/10.1145/3459664
20. Fathollahi-Fard, A. M., Woodward, L., & Akhrif, O. (2024). A scenario-based robust optimization model for the sustainable distributed permutation flow-shop scheduling problem. Annals of Operations Research, 1-42. https://doi.org/10.1007/s10479-024-05940-7
21. Pan, Z., Wang, L., Wang, J., & Lu, J. (2021). Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling. IEEE Transactions on Emerging Topics in Computational Intelligence, 7(4), 983-994. https://doi.org/10.1109/tetci.2021.3098354
22. Framinan, J. M., Gupta, J. N., & Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. Journal of the Operational Research Society, 55(12), 1243-1255. https://doi.org/10.1057/palgrave.jors.2601784
23. 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.3800010110
24. Taillard, E. (1993). Benchmarks for basic scheduling problems. european journal of operational research, 64(2), 278-285. https://doi.org/10.1016/0377-2217(93)90182-m
25. Miljus, R. C., Campbell, J. P., Dunnette, M. D., Lawler, E. E., & Weick, K. E. (1971). Managerial Behavior, Performance, and Effectiveness. Industrial and Labor Relations Review, 24(3), 487. https://doi.org/10.2307/2521540
26. Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. In Handbook of metaheuristics (pp. 320-353). Boston, MA: Springer US. https://doi.org/10.1007/0-306-48056-5_11
27. Osman, I. H., & Potts, C. N. (1989). Simulated annealing for permutation flow-shop scheduling. Omega, 17(6), 551-557. https://doi.org/10.1016/0305-0483(89)90059-5
28. Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. Computers & operations research, 22(1), 5-13. https://doi.org/10.1016/0305-0548(93)e0014-k
29. Nowicki, E., & Smutnicki, C. (2005). An advanced tabu search algorithm for the job shop problem. Journal of Scheduling, 8(2), 145-159. https://doi.org/10.1007/s10951-005-6364-5
30. Rajendran, C., & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research, 155(2), 426-438. https://doi.org/10.1016/s0377-2217(02)00908-6
31. Tasgetiren, M. F., Liang, Y. C., Sevkli, M., & Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European journal of operational research, 177(3), 1930-1947. https://doi.org/10.1016/j.ejor.2005.12.024
32. Qian, W. (2008). Adaptive differential evolution algorithm for multiobjective optimization problems. Applied Mathematics and Computation, 201(1-2), 431-440. https://doi.org/10.1016/j.amc.2007.12.052


[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]

33. Dong, L., Tian, D., Lu, W., & Liu, Z. (2020). Estimating the efficient parameter values of different neighborhood search techniques of simulated annealing in forest spatial planning problems. IEEE Access, 8, 115905-115921. https://doi.org/10.1109/access.2020.3004563
34. Lin, S. W., Ying, K. C., & Huang, C. Y. (2013). Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. International Journal of Production Research, 51(16), 5029-5038. https://doi.org/10.1080/00207543.2013.790571
35. Framinan, J. M., Perez-Gonzalez, P., & Fernandez-Viagas, V. (2023). An overview on the use of operations research in additive manufacturing. Annals of Operations Research, 322(1), 5-40. https://doi.org/10.1007/s10479-022-05040-4
36. Zeng, D., Zhan, J., Peng, W., & Zeng, Z. (2023). Evolutionary job scheduling with optimized population by deep reinforcement learning. Engineering Optimization, 55(3), 494-509. https://doi.org/10.1080/0305215x.2021.2013479
37. Tang, H., & Dong, J. (2024). Solving flexible job-shop scheduling problem with heterogeneous graph neural network based on relation and deep reinforcement learning. Machines, 12(8), 584. https://doi.org/10.1007/s00170-023-11707-4
38. Patel, P., Gopal, K. M., Patel, N. C., Sahu, B. K., Tripathy, S. P., Altayeb, M., ... & Touti, E. (2026). Fuzzy adaptive human memory optimization based optimal task scheduling in cloud computing. Scientific Reports. https://doi.org/10.1038/s41598-026-51611-x
39. Chang, X., Jia, X., & Ren, J. (2025). A reinforcement learning enhanced memetic algorithm for multi-objective flexible job shop scheduling toward Industry 5.0. International Journal of Production Research, 63(1), 119-147. https://doi.org/10.1080/00207543.2024.2357740
40. Mojumder, M. U., & Nuruzzaman, M. (2025). AI-driven optimization of warehouse layout and material handling: A quantitative study on efficiency and space utilization. Review of Applied Science and Technology, 4(02), 233-273. https://doi.org/10.63125/bgxb1z53
41. Jalali Khalil Abadi, Z., Mansouri, N., & Javidi, M. M. (2024). Deep reinforcement learning-based scheduling in distributed systems: a critical review. Knowledge and Information Systems, 66(10), 5709-5782. https://doi.org/10.1007/s10115-024-02167-7

[ Downloaded from ijaor.com on 2026-11-24 ]

[ DOI: 10.71885/ijorlu-2026-3-731 ]