

Article

BNMG: A Novel Deterministic Hybrid Algorithm with Global Makespan-Based Swap Mechanism for the Permutation Flow Shop Scheduling Problem

Yilmaz Ar ¹  and Nermin Kartli ^{2,*} ¹ Department of Computer Engineering, Ankara University, Ankara 06830, Türkiye; ar@ankara.edu.tr² Department of Computer Sciences, Ankara University, Ankara 06100, Türkiye

* Correspondence: nkartli@ankara.edu.tr

Abstract

The Permutation Flow Shop Planning Problem (PFSSP) is fundamental and frequently encountered in manufacturing systems and service operations. This problem is known to be NP-hard for three or more machines. Therefore, various heuristic and metaheuristic algorithms exist to approximate solutions to the problem. The solutions produced by deterministic heuristic algorithms are frequently used as initial solutions for population-based metaheuristic algorithms because they provide feasible schedules of relatively high quality within a short computational time. Heuristic algorithms are also divided into two groups: deterministic and random. In this study, we aim to develop a new deterministic method that improves both solution quality and computational efficiency. Rather than replacing existing deterministic heuristics, we propose a method that aims to enrich the design space of deterministic PFSSP heuristics by introducing two new problem-specific sequence improvement operators inspired by classical sorting principles. The proposed method is based on the integrated use of three complementary components: (i) a Bubble-Swap-based neighborhood structure that increases local search power, (ii) an NEH-style insertion mechanism that uses the strong insertion logic of the classical NEH algorithm, and (iii) a Merge-Global-Swap strategy that provides global optimization based on the completion time value of the entire sequence at each merge step. By integrating these three components, we develop a new deterministic hybrid algorithm called BNMG (Bubble-NEH-Merge-Global). We also call the locally search-enhanced version of our algorithm BNMG-II. Furthermore, we propose a new metric that accounts for computation time to evaluate the performance of the algorithms. When we comprehensively compare the BNMG and BNMG-II algorithms with the classical NEH, the recently developed vN-NEH and NEH-II, vN-NEH+ algorithms in Taillard test problems, we report that they exhibit superior performance according to the mean relative deviation metric (M1/ARPD) and the proposed new metric.



Academic Editor: Tamas Kiss

Received: 20 June 2026

Revised: 17 July 2026

Accepted: 24 July 2026

Published: 29 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: heuristic algorithm; makespan minimization; meta-heuristic algorithm; NEH algorithm; scheduling problem

1. Introduction

Heuristic, metaheuristic, and hybrid algorithms are successfully applied to path planning [1,2], collaborative filtering [3,4], supply chain management [5–7], various prediction problems [8], community detection [9–11], and other applications. One of these broad application areas is scheduling problems.

Scheduling problems are one of the most fundamental and intensively studied problem classes in operations research and combinatorial optimization. These problems are generally divided into two main groups: job/machine scheduling and project scheduling. In job/machine scheduling problems, the aim is to allocate a limited number of resources as efficiently as possible under specific process sequences and technological constraints. Processor scheduling applications in production lines and computer systems can be given as examples of this class. In project scheduling problems, the aim is to optimize the project completion time or total cost by considering the priority relationships between activities and resource constraints.

One of the most commonly studied problem classes in the production planning literature is the Flow Shop Scheduling Problem (FSSP) [12–16]. The fact that this problem is directly related to real-world applications such as production systems, assembly lines, and service operations increases its importance both academically and industrially.

The Permutation Flow Shop Scheduling Problem (PFSSP) is a special case of the FSSP in which all jobs follow the same processing order across the machines. The most common objective function in this problem is to minimize the total completion time (makespan), which represents the time it takes for the last job to complete on the last machine. PFSSP is an NP-hard problem for three or more machines, and the exponential growth of the solution space as the problem size increases significantly limits the applicability of exact solution methods. Therefore, the literature focuses on developing heuristic and metaheuristic methods that can produce high-quality solutions within acceptable computation times [17].

Among deterministic heuristic methods, the Nawaz–Enscore–Ham (NEH) algorithm [18] is considered one of the most effective approaches for PFSSP. The NEH algorithm constructs the solution incrementally by sorting jobs according to their total processing times and then placing each job in the most optimal position within the partial sequence. Despite its simple structure, the NEH algorithm has formed the basis of many studies in the literature due to its successful balance between solution quality and computation time.

In recent years, approaches incorporating rich neighborhood structures and additional local search strategies have been proposed to make the NEH algorithm more flexible. In this context, the vN-NEH and vN-NEH+ algorithms developed by [19] are noteworthy for integrating variable neighborhood structures into the fitting phase. Such approaches demonstrate that a wider solution space can be effectively explored within a deterministic framework.

Despite the widespread use of metaheuristic and stochastic methods, deterministic heuristic algorithms continue to play an important role in the scheduling literature. Their main advantages include full reproducibility, the absence of parameter tuning, low computational cost, and the ability to generate high-quality feasible schedules in a deterministic manner. These characteristics make deterministic heuristics particularly attractive for large-scale scheduling problems and real-time decision-making environments. Moreover, deterministic heuristics are frequently employed as standalone solution methods and are also commonly adopted as initialization procedures or components of hybrid optimization frameworks. However, the effectiveness of a particular deterministic heuristic as an initialization strategy depends on the characteristics of the metaheuristic in which it is embedded. Therefore, in this study, the proposed BNMG framework is evaluated solely as a standalone deterministic heuristic, while its integration into metaheuristic frameworks is left as future work. Consequently, developing new deterministic search operators and improving the solution quality of deterministic heuristics remain important research directions in the PFSSP literature.

In this study, a new deterministic hybrid algorithm called BNMG (Bubble-NEH-Merge-Global) is proposed for PFSSP. The proposed method is based on the

integrated use of three complementary components: (i) a Bubble-Swap operator that provides local improvement by swapping between neighboring jobs, (ii) an NEH-like insertion mechanism that produces high-quality partial solutions, and (iii) a Merge-Global-Swap approach that makes decisions by considering the makespan value of the entire sequence in each merging step.

Additionally, an extended BNMG-II algorithm with an iterative improvement step is also presented. The most important feature of the proposed Merge-Global-Swap mechanism is that decisions are made not only based on local sub-arrays, but also considering the performance of the entire work array. In this respect, the method differs from the local decision mechanisms commonly used in classical NEH and its derivatives. Furthermore, the Bubble-Swap operator is systematically considered as a deterministic neighborhood structure for PFSSP.

The main contributions of this study are summarized as follows:

- A new deterministic constructive heuristic, BNMG, is proposed for the PFSSP. Unlike previous NEH-based heuristics, BNMG introduces two problem-specific search operators inspired by the principles of Bubble Sort and Merge Sort. To the best of our knowledge, these sorting concepts have not previously been adapted as optimization operators for the PFSSP.
- Rather than serving merely as an initialization mechanism, BNMG is designed as a standalone deterministic scheduling algorithm that directly improves solution quality while preserving the computational efficiency of the classical NEH heuristic.
- The proposed Merge-Global-Swap mechanism performs global sequence refinement by evaluating the makespan after each merge stage rather than relying solely on adjacent insertion-based improvements.
- An enhanced version, BNMG-II, incorporates an additional deterministic local improvement phase, providing further solution improvements without introducing stochastic components or parameter tuning.
- The proposed algorithms are extensively evaluated on all 120 Taillard benchmark instances and compared with five representative deterministic heuristics using both solution-quality and time-aware performance metrics. Additional runtime and statistical analyses are also provided to support the experimental findings.

Accordingly, the objective of this study is not to develop an initialization strategy for metaheuristic algorithms, but to design a competitive standalone deterministic heuristic that improves the performance of the classical NEH framework while preserving its computational efficiency.

The rest of the article is organized as follows: Section 2 presents the relevant literature. Section 3 provides the problem definition and basic representations. We describe the proposed algorithm in Section 4. Section 5 includes the experimental results and analyses. Finally, we discuss the results of the study in Section 6.

2. Related Work

The Permutation Flow Scheduling Problem (PFSSP) is a significant problem both theoretically and due to its applications in production and service systems. This problem has been extensively studied by researchers for many years. It is known that the problem is NP-hard for three or more machines. Therefore, exact solution methods such as branching and bounding, dynamic programming, and mixed integer programming are only efficient for small-scale problems. This situation has made the development of heuristic and metaheuristic methods for finding approximate solutions attractive to researchers.

As mentioned above, the Nawaz–Enscore–Ham (NEH) algorithm is one of the most efficient methods for PFSSP. The NEH algorithm iteratively generates the solution by

sorting jobs according to their total processing times and then placing each job in the optimal position within a partial sequence.

Subsequent studies have attempted to improve NEH through various strategies. Fernandez-Viagas and Framinan [20] proposed the Fast NEH (FNEH) algorithm, which reduces computational time without significantly affecting solution quality. Rajendran [21] developed a heuristic algorithm for scheduling that aims to minimize the total flow time of jobs in a flow shop. In this algorithm, the author developed a heuristic preference relation and used it as the basis for job insertion. Dong et al. [22] and Kalczynski and Kamburowski [23] developed improved versions of the NEH algorithm, which refines the initial NEH sequence by selectively interchanging jobs with large total processing times if the swap reduces the makespan. In this study, we used Dong et al.'s refinement for comparison and used the abbreviation NEH-II for this algorithm throughout. Puka et al. [19,24] developed N-NEH+, vN-NEH, and VN-NEH+ algorithms as modifications of the NEH algorithm. Other studies explored modifications to the job ordering phase [25,26], insertion rules [27], greedy approach [28,29] or local search integration [30,31].

Machine-based heuristics aim to incorporate information about machine workloads and idle times into the sequencing process. Early studies employed load-based priority rules, such as those minimizing machine utilization variance or balancing processing times among machines [32–34]. These studies combined machine load metrics with constructive insertion rules to achieve better initial sequences.

Another line of research focuses on block-based heuristics, where groups of jobs (blocks) are inserted or rearranged together rather than individually. Block insertion and pairwise evaluation techniques have been shown to accelerate convergence and improve local sequencing decisions [35–37].

The incorporation of local improvement phases—particularly those inspired by NEH-II—has proven effective in refining heuristic schedules. The NEH-II procedure performs selective job interchanges among the most time-consuming jobs, improving the global sequence without significant computational overhead. Inspired by this, several NEH hybrids have adopted similar post-optimization phases [26].

Metaheuristic algorithms such as genetic algorithms (GA), teaching–learning-based optimization (TLBO), particle swarm optimization (PSO), and simulated annealing (SA) have also been extensively applied to PFSSP. Recent studies emphasize hybridization—e.g., GA-NEH, memetic algorithms, and TLBO with NEH initialization—to improve convergence speed and solution quality [38].

In [39], a hybrid Grey Wolf Optimization (HGWO) approach is proposed. First, a cooperative initialization strategy based on the tent chaotic map and the improved Nawaz-Enscore-Ham (NEH) method is presented to improve the quality of the initial solution. Second, a Lévy flight strategy is integrated to balance the exploration and exploitation processes of the problem features. Third, a critical block replacement mechanism based on the critical path approach is proposed along with crossover and mutation strategies to avoid getting stuck in local optima.

Fazekas et al. [40] propose a novel combination of two powerful heuristics previously developed by the authors—the discrete bacterial memetic evolution algorithm and a suitably adapted Monte Carlo tree search approach. They develop a hybrid algorithm by combining these two methods. The authors evaluated the proposed algorithm on the Taillard benchmark dataset.

In the study [41], to monitor current trends in the field and to more comprehensively examine the improvement capabilities of Particle Swarm Optimization (PSO), the standard PSO was first developed. This structure was then adapted to solve Permutation Flow Shop Scheduling Problems (PFSP) and hybridized with Variable Neighborhood Search (PSO-

VNS) and Simulated Annealing (PSO-VNS-SA). The effectiveness of the hybridization was verified through internal comparisons conducted by Taillard on 120 examples generated with different problem sizes. Furthermore, additional comparisons with other hybrid metaheuristic methods reported in the literature revealed that the hybrid PSO (HPSO) developed in this study exhibits exceptional performance. The low ARPD (Average Relative Performance Difference) value of 0.48 achieved by the algorithm demonstrates its significantly superior performance compared to other methods in both structural robustness and production time improvement.

Khurshid et al. [42] propose an Improved Evolution Strategy to reduce the total completion time of the PFSSP. Two variants of this strategy, ES5 and ES10, have been developed. The initial solution is based on the shortest processing time rule. The ES5 approach produces four offspring from a single parent, while the ES10 approach produces nine. The selection pool is defined to consist of both the parents and offspring. To minimize computational time and perform an efficient search in the solution space, a quadruple swap mutation operator is proposed. Furthermore, a variable mutation rate is used to improve the results, gradually decreasing the mutation rate as the number of iterations increases.

Mishra and Shrivastava [43] aim to investigate the performance of the Jaya algorithm on the permutation flow shop scheduling problem. First, to adapt the Jaya algorithm to this problem, a random priority value is assigned to each job in the permutation sequence. Second, the generated job priority vector is transformed into a job permutation vector using the Maximum Rank Value rule. Mishra and Shrivastava [44] proposed a Hybrid Iterative Greedy-based Jaya algorithm (HIGJ). The proposed approach integrates the population-based optimization structure of the Jaya algorithm with the single-solution-based Iterative Greedy (IG) algorithm to ensure that population members arrive at better solutions. The primary objective is to minimize the total completion time. The destruction and construction phases of the Iterative Greedy algorithm are integrated into the optimal solution of the Jaya algorithm. Furthermore, a local search method is incorporated into the algorithm to improve the overall quality of candidate solutions and enable faster convergence to the optimal solution. To evaluate the effectiveness of the proposed HIGJ algorithm, a comprehensive comparative study and statistical analysis are conducted using publicly available benchmarks.

In the study [45], three metaheuristic optimization algorithms based on discrete differential evolution (DE) are implemented to minimize the total completion time in PFS scheduling problems. To this end, three different optimization approaches based on Discrete Differential Evolution and Discrete Self-Adaptive Differential Evolution (DSDE) are proposed, namely DDE-PFS, DSADE-PFS1, and DSADE-PFS2, respectively. Numerical benchmarks presented by Carlier (CB), Heller (HB), Reeves (RB), and Taillard (TB) are used to evaluate the effectiveness of the proposed algorithms. The performances of the optimization methods are comprehensively analyzed using relative percentage error (RPE), convergence behavior, standard deviation (Std), and Friedman and post hoc Nemenyi statistical tests.

The studies [46,47] propose an improved version of the tabu search algorithm to solve the problem.

Existing studies in the literature reveal that most proposed methods either focus on improving NEH-based structures through a specific component or resort to meta-heuristic hybrid approaches to achieve strong performance. However, an integrated structure that enhances local search power, preserves NEH's effective insertion mechanism, and enables global decision-making across the entire sequence within a deterministic framework has been addressed to a limited extent. In this study, we aim to fill this gap by combining Bubble-Swap-based local improvement, NEH-style insertion, and Merge-Global-Swap global

decision-making mechanisms under a single deterministic hybrid structure. Furthermore, in contrast to performance evaluations that mostly focus solely on solution quality in the literature, we propose a new metric that considers both solution quality and computation time, thus providing a different perspective on the evaluation process.

Although Bubble Sort and Merge Sort are among the most fundamental comparison-based sorting algorithms in computer science, their underlying principles have not previously been adapted as deterministic neighborhood generation mechanisms for the PFSSP. This study explores this unexplored direction by redesigning these sorting concepts as makespan-oriented sequence improvement operators.

3. Preliminaries

3.1. Mathematical Model of the PFSSP

The Permutation Flow Shop Scheduling Problem (PFSSP) is a scheduling problem that requires n jobs to be processed in the same order on m machines. The goal is to minimize the makespan value, which is the time it takes for the last job to be completed on the last machine.

Some notation for the mathematical model of the problem is included.

3.1.1. Notations

$J = \{1, 2, \dots, n\}$: Set of jobs.

$M = \{1, 2, \dots, m\}$: Set of machines.

$p_{j,k}$: Processing time of job j on machine k .

$C_{j,k}$: Completion time of job j on machine k .

$\pi = (\pi_1, \pi_2, \dots, \pi_n)$: Permutation of all jobs on all machines, where π_i represents the job processed at position i in the schedule.

$C_{\max} = C_{\pi_n, m}$, which is called as makespan.

3.1.2. Definition of the Problem

In this study, we address the following problem:

$$\min C_{\max} \quad (1)$$

under the constraints

$$C_{j,k} \geq C_{j,k-1} + p_{j,k} \quad \forall j \in J, k = 2, \dots, m \quad (2)$$

$$C_{\pi_i, k} \geq C_{\pi_{i-1}, k} + p_{\pi_i, k} \quad \forall i = 2, \dots, n, k \in M \quad (3)$$

$$C_{\pi_1, 1} = p_{\pi_1, 1} \quad (4)$$

$$C_{\pi_i, k} \geq \max(C_{\pi_i, k-1}, C_{\pi_{i-1}, k}) + p_{\pi_i, k} \quad (5)$$

$$\forall i = 2, \dots, n, k = 2, \dots, m$$

Here, (2) is a flow-type processing constraint, which means that each job must be processed in a fixed order on the machines; (3) is a machine capacity constraint, showing that each machine can process only one job at a time; and (4) represents the start time constraint. Constraint (5) recursively defines the completion time of each job on each machine. Since the first job and the first machine are separately initialized by Constraint (4), Equation (5) is applied only for $i = 2, \dots, n$ and $k = 2, \dots, m$.

3.2. NEH Algorithm

Algorithm 1 provides a standard implementation of the classical NEH heuristic, generating a sequence of jobs by incrementally placing jobs in positions that minimize their completion times. Owing to its simplicity, computational efficiency, and consistently competitive solution quality, the NEH algorithm has remained one of the most influential deterministic heuristics for the PFSSP. Therefore, it is frequently used as a benchmark in evaluating the performance of new methods proposed in the literature.

Algorithm 1 NEH Algorithm (Nawaz–Enscore–Ham)

Require: n jobs, m machines, processing times p_{ij}

Ensure: Job permutation π

1: Compute total processing time for each job:

$$T_i = \sum_{j=1}^m p_{ij}, \quad i = 1, \dots, n$$

2: Sort jobs in decreasing order of T_i

3: Let the sorted list be $\langle j_1, j_2, \dots, j_n \rangle$

4: Evaluate sequences (j_1, j_2) and (j_2, j_1)

5: Set π to the sequence with smaller makespan

6: **for** $k = 3$ to n **do**

7: $best_C_{\max} \leftarrow \infty$

8: **for** $r = 1$ to k **do**

9: Insert job j_k at position r in π

10: Compute makespan $C_{\max}$

11: **if** $C_{\max} < best_C_{\max}$ **then**

12: $best_C_{\max} \leftarrow C_{\max}$

13: $\pi^* \leftarrow$ current sequence

14: **end if**

15: **end for**

16: $\pi \leftarrow \pi^*$

17: **end for**

18: **return** π

3.3. NEH-II Algorithm (Iterative Improvement NEH)

In the literature, applying an additional local optimization (post-optimization) phase to the initial permutation has become a common approach to improve the solution quality of the NEH algorithm. In such methods, the initial sequence generated by NEH is not considered fixed; instead, each job is sequentially removed from its current position and repositioned to all possible positions, and the completion times of the resulting permutations are compared. If a better solution is found, the new permutation is accepted, and the process is repeated until no improvement remains. This approach can provide significant improvements in solution quality by creating an additional local search layer for NEH-based heuristics [48,49].

In this study, this approach is called “NEH-II”. Here, the “II” represents the iterative improvement phase applied following the constructive NEH phase of the algorithm. The NEH-II algorithm consists of two main phases:

- Constructor phase: An initial job sequence is generated using the standard NEH algorithm.
- Refining phase: An iterative insertion-based local search is applied to the resulting permutation, where each job is repositioned to all possible locations.

The NEH-II algorithm is summarized in Algorithm 2.

Algorithm 2 NEH-II Algorithm**Require:** Processing time matrix $P_{n \times m}$ **Ensure:** Improved job sequence S^* minimizing makespan

1: Compute total processing time for each job:

$$T_i = \sum_{j=1}^m P_{ij}, \quad i = 1, \dots, n$$

2: Sort jobs in decreasing order of T_i 3: Let the ordered list be $\langle j_1, j_2, \dots, j_n \rangle$ 4: Evaluate sequences (j_1, j_2) and (j_2, j_1) 5: Select the better one as initial sequence S 6: **for** $k = 3$ to n **do**7: **for** $r = 1$ to $|S| + 1$ **do**8: Temporarily insert job j_k at position r in S

9: Compute the corresponding makespan

10: **end for**11: Fix job j_k at the position yielding minimum makespan12: **end for**13: $S^* \leftarrow S$ 14: improved $\leftarrow$ true15: **while** improved **do**16: improved $\leftarrow$ false17: **for** each job $i \in S^*$ **do**18: **for** each position $pos \in \{1, \dots, |S^*|\}$ **do**19: Remove job i from its current position20: Insert job i at position pos

21: Compute the makespan of the new sequence

22: **if** makespan $<$ best_makespan **then**23: Update S^* and best_makespan24: improved $\leftarrow$ true25: **end if**26: **end for**27: **end for**28: **end while**29: **return** S^* 3.4. *vN-NEH*

vN-NEH adds a variable neighborhood mechanism based on adjacent swaps to the classic NEH structure. *vN-NEH+* strengthens this structure with iterative re-insertion similar to NEH-II. Both algorithms rely on local decision mechanisms; swap or insertion operations are evaluated via partial sequences.

3.5. *vN-NEH+*

The Algorithm 3 aims to improve solution quality by adding a variable neighborhood mechanism based on neighbor swapping to the classical NEH constructor heuristic. However, this structure is limited to local swap improvements. Therefore, an extended version of the *vN-NEH* algorithm with an additional re-insertion-based local search phase has been proposed in the literature to further improve solution quality. This extended version is called *vN-NEH+*.

In the Algorithm 4, the initial permutation is first obtained by running the *vN-NEH* procedure, then each job is removed from its current position and repositioned to all possible positions within the permutation. The completion time of each new permutation is calculated, and only changes that provide improvement are accepted. This process continues iteratively until no improvement remains.

Algorithm 3 vN-NEH Algorithm**Require:** Processing time matrix P with n jobs and m machines**Ensure:** Job sequence π minimizing makespan

```

1: Compute total processing time  $T_j = \sum_{k=1}^m P_{j,k}$  for each job  $j$ 
2: Sort jobs in decreasing order of  $T_j$  to obtain list  $L$ 
3: Initialize  $\pi \leftarrow [L_1]$ 
4: for  $i = 2$  to  $n$  do
5:    $job \leftarrow L_i$ 
6:    $\pi^{best} \leftarrow \emptyset, C^{best} \leftarrow \infty$ 
7:   for  $pos = 0$  to  $|\pi|$  do
8:     Insert  $job$  into  $\pi$  at position  $pos$  to obtain  $\pi'$ 
9:     Compute makespan  $C_{max}(\pi')$ 
10:    if  $C_{max}(\pi') < C^{best}$  then
11:       $\pi^{best} \leftarrow \pi'$ 
12:       $C^{best} \leftarrow C_{max}(\pi')$ 
13:    end if
14:  end for
15:   $\pi \leftarrow \pi^{best}$ 
16:  for all adjacent pairs  $(i, i + 1)$  in  $\pi$  do
17:    Swap jobs at positions  $i$  and  $i + 1$  to obtain  $\pi''$ 
18:    if  $C_{max}(\pi'') < C_{max}(\pi)$  then
19:       $\pi \leftarrow \pi''$ 
20:    end if
21:  end for
22: end for
23: return  $\pi$ 

```

▷ Variable Neighborhood swap improvement

Algorithm 4 vN-NEH+ Algorithm**Require:** Processing time matrix P **Ensure:** Improved job sequence π

```

1:  $\pi \leftarrow \text{vN-NEH}(P)$ 
2: repeat
3:    $\text{improved} \leftarrow \text{false}$ 
4:   for  $i = 1$  to  $|\pi|$  do
5:     Remove job  $\pi_i$  from sequence to obtain  $\pi^-$ 
6:      $best\_local \leftarrow \pi, C^{best} \leftarrow C_{max}(\pi)$ 
7:     for  $pos = 0$  to  $|\pi^-|$  do
8:       Insert  $\pi_i$  into  $\pi^-$  at position  $pos$  to obtain  $\pi'$ 
9:       Compute  $C_{max}(\pi')$ 
10:      if  $C_{max}(\pi') < C^{best}$  then
11:         $best\_local \leftarrow \pi'$ 
12:         $C^{best} \leftarrow C_{max}(\pi')$ 
13:      end if
14:    end for
15:     $\pi \leftarrow best\_local$ 
16:  end for
17: until  $\text{improved} = \text{false}$ 
18: return  $\pi$ 

```

4. Proposed Methodology**4.1. General Structure of the BNMG Algorithm**

In this study, a new deterministic heuristic algorithm, called BNMG, is proposed for solving the PFSSP. Rather than modifying only the insertion mechanism of the classical NEH heuristic, BNMG integrates three complementary deterministic search operators:

NEH-based constructive insertion, Bubble-Swap local refinement, and Merge-Global-Swap sequence optimization. These operators work cooperatively to improve schedule quality while maintaining the low computational complexity, deterministic behavior, and reproducibility of the algorithm.

The proposed algorithm is named **BNMG** (Bubble–NEH–Merge–Global) and consists of four basic stages:

1. **Generating the initial solution:** In the first stage, jobs are sorted according to their total processing times in descending order.
2. **Local optimizing with Bubble-Swap:** A deterministic local search operator, inspired by bubble sort logic based on swapping neighboring job pairs, is applied to the obtained initial permutation. This operator only accepts swaps that improve completion time.
3. **Reconstructive repositioning:** The job order obtained after the Bubble-Swap phase is re-evaluated using the NEH algorithm's placement procedure.
4. **Merge-based global optimization:** In the final stage, the permutation is recursively divided into sub-sequences, and the effect of job pair swapping on the entire permutation during the merge process is evaluated.

We give the general flow of the algorithm in Algorithm 5. We describe the procedures Bubble-Swap and Merge-Global in detail in Section 4.2 and Section 4.3, respectively.

Algorithm 5 BNMG Algorithm

Require: Job set J , processing time matrix P

Ensure: Final job permutation π^*

- 1: $\pi \leftarrow \text{DESCENDING_ORDER}(J, P)$
 - 2: $\pi \leftarrow \text{BUBBLE-SWAP}(\pi, P)$
 - 3: $\pi \leftarrow \text{NEH_CONSTRUCT}(\pi, P)$
 - 4: $\pi^* \leftarrow \text{MERGEGLOBAL}(\pi, P)$
 - 5: **return** π^*
-

4.2. Bubble-Swap Operator

The second stage of the BNMG algorithm proposed in this study is the Bubble-Swap operator, a deterministic local search operator based on the swapping of adjacent pairs of jobs. While inspired by the classical adjacent swap adjacency structure, this operator follows the structural features of the bubble sort algorithm in terms of its search strategy.

The Bubble-Swap operator iterates from left to right on a given initial permutation, testing only the swapping of adjacent pairs of jobs. After each swap attempt, the completion time of the resulting new permutation is calculated, and only swaps that provide improvement are accepted. When a pass is completed, similar to the bubble sort algorithm, an element at the end of the sequence is considered “fixed” and is not re-evaluated in subsequent passes. This mechanism prevents unnecessary repetitions by enabling a directional search in the permutation space.

The primary purpose of the Bubble-Swap operator is to improve the quality of the solution by making low-cost local changes to the order obtained after the NEH_order stage. This results in a more suitable starting sequence for the subsequent constructive NEH_construct stage.

The pseudocode for the algorithm is given in Algorithm 6.

In the algorithm, the outer loop ensures that an element from the end of the array is fixed in each pass, while the inner loop only tests neighbor swaps on the part that has not yet been fixed. This structure, unlike classical neighbor-swap-based local searches, leads to a gradual stabilization towards the end of the permutation and ensures a more orderly progress of the search process.

Algorithm 6 Bubble-Swap Operator**Require:** Initial permutation π , processing time matrix P **Ensure:** Locally improved permutation π

```

1:  $n \leftarrow |\pi|$ 
2: for  $i \leftarrow 1$  to  $n - 1$  do
3:   for  $j \leftarrow 1$  to  $n - i$  do
4:      $\pi' \leftarrow \pi$ 
5:     swap positions  $j$  and  $j + 1$  in  $\pi'$ 
6:     if  $\text{MAKESPAN}(\pi', P) < \text{MAKESPAN}(\pi, P)$  then
7:        $\pi \leftarrow \pi'$ 
8:     end if
9:   end for
10: end for
11: return  $\pi$ 

```

4.3. Merge-Based Global Swap Operator

In the final stage of the BNMG algorithm, a global optimization operator based on a recursive divide-and-combine structure is proposed to evaluate larger-scale permutation changes in the solution space. This operator is inspired by the structural framework of the classical merge sort algorithm, but makes comparison and merging decisions not only based on local key values but also on the effect on the completion time of the entire permutation.

In the classical merge sort algorithm, when merging two sub-arrays, the order of the elements is determined by a single comparison key. In the proposed method, two jobs at the beginning of the left and right sub-arrays are selected in the merging stage, and the new completion time that will be obtained if the positions of these jobs within the entire permutation are swapped is calculated. If this change reduces the global completion time, the relevant swap is applied permanently. Therefore, the proposed operator can evaluate not only local neighborhood changes but also the interaction of jobs in distant positions of the permutation.

The primary purpose of the merge-based global swap operator is to reduce the likelihood of getting stuck in local minima by systematically examining larger-scale ordering changes on the solution obtained in the Bubble-Swap phase and reconstructed with NEH_construct.

The pseudocode for the algorithm is given in Algorithm 7.

The proposed operator differs from the classical merge sort algorithm in three key points. It is separated as follows:

- The completion time of the entire permutation is used instead of local key values as the comparison criterion.
- During merging, not only is a sorting decision made, but persistent swap operations can also be applied to the global permutation.
- The algorithm is used not for sorting purposes, but to improve an existing job queue.

4.4. BNMG-II Algorithm

The BNMG algorithm is based on improving an initial constructive solution using local and global optimization operators. However, in the literature, improved versions that include an additional local search phase after the constructive phase have also been proposed to improve the performance of the NEH algorithm. Such approaches are generally called *NEH-II*, and the completion time is further reduced by applying iterative substitution or repositioning operations on the obtained solution.

Algorithm 7 Merge-Based Global Swap Operator**Require:** Permutation π , processing time matrix P **Ensure:** Globally refined permutation π

```

1: if  $|\pi| \leq 1$  then
2:   return  $\pi$ 
3: end if
4: Split  $\pi$  into two halves:  $\pi_L$  and  $\pi_R$ 
5:  $\pi_L \leftarrow \text{MERGEGLOBAL}(\pi_L, P)$ 
6:  $\pi_R \leftarrow \text{MERGEGLOBAL}(\pi_R, P)$ 
7:  $\pi_M \leftarrow$  empty list
8: while  $\pi_L \neq \emptyset$  and  $\pi_R \neq \emptyset$  do
9:    $a \leftarrow$  first element of  $\pi_L$ 
10:   $b \leftarrow$  first element of  $\pi_R$ 
11:   $\pi' \leftarrow \pi$ 
12:  swap positions of  $a$  and  $b$  in  $\pi'$ 
13:  if  $\text{MAKESPAN}(\pi', P) < \text{MAKESPAN}(\pi, P)$  then
14:     $\pi \leftarrow \pi'$ 
15:  end if
16:  if position of  $a$  precedes position of  $b$  in  $\pi$  then
17:    append  $a$  to  $\pi_M$ 
18:    remove first element from  $\pi_L$ 
19:  else
20:    append  $b$  to  $\pi_M$ 
21:    remove first element from  $\pi_R$ 
22:  end if
23: end while
24: append remaining elements of  $\pi_L$  to  $\pi_M$ 
25: append remaining elements of  $\pi_R$  to  $\pi_M$ 
26: return  $\pi_M$ 

```

In this study, a similarly extended version of the proposed BNMG algorithm is also considered and named BNMG-II. The BNMG-II algorithm takes the permutation obtained by BNMG as the starting point and applies an iterative local search process on this permutation. Thus, the global optimizations provided by BNMG are combined with NEH-II-style fine-tuning mechanisms.

The working steps of the BNMG-II algorithm can be summarized as follows:

1. Jobs are sorted in descending order according to total processing times.
2. The Bubble-Swap operator is applied to the resulting order.
3. The updated order is constructively repositioned using the NEH_construct procedure.
4. The resulting permutation is improved with a Merge-based global swap operator.
5. In the final stage, iterative local search is applied to the resulting permutation to test for neighbor and distant position changes, and only changes that reduce completion time are accepted.

We give the pseudocode for the BNMG-II algorithm in Algorithm 8.

The iterative local search procedure used in the final stage of the BNMG-II algorithm systematically evaluates the effect of moving each job to different positions within the permutation on the completion time. This process continues until an iteration where no improvement is achieved; thus, the algorithm searches until it reaches a local optimum under the given neighborhood structure.

The BNMG-II algorithm follows the optimization framework of the classical NEH-II heuristic while replacing its original job ordering with the deterministic schedule generated by BNMG. Since this schedule has already been refined through the Bubble-Swap and Merge-Global-Swap operators, BNMG-II starts its insertion process from a higher-quality

deterministic sequence. This design enables BNMG-II to further improve schedule quality while preserving the deterministic nature and computational efficiency of the original NEH-II framework.

Algorithm 8 BNMG-II Algorithm

Require: Job set J , processing time matrix P

Ensure: Final improved permutation π^*

```

1:  $\pi \leftarrow \text{DESCENDINGORDER}(J, P)$ 
2:  $\pi \leftarrow \text{BUBBLE-SWAP}(\pi, P)$ 
3:  $\pi \leftarrow \text{NEHCONSTRUCT}(\pi, P)$ 
4:  $\pi \leftarrow \text{MERGEGLOBAL}(\pi, P)$ 
5: improved  $\leftarrow$  true
6: while improved do
7:   improved  $\leftarrow$  false
8:   for each job  $i$  in  $\pi$  do
9:     for each position  $pos$  in  $\pi$  do
10:       $\pi' \leftarrow$  permutation obtained by moving job  $i$  to position  $pos$ 
11:      if  $\text{MAKESPAN}(\pi', P) < \text{MAKESPAN}(\pi, P)$  then
12:         $\pi \leftarrow \pi'$ 
13:        improved  $\leftarrow$  true
14:      end if
15:    end for
16:  end for
17: end while
18:  $\pi^* \leftarrow \pi$ 
19: return  $\pi^*$ 

```

Termination and Deterministic Behavior

Unlike iterative metaheuristics, the proposed BNMG algorithm follows a deterministic constructive procedure consisting of a finite sequence of predefined operations. Starting from the initial job ordering, the algorithm successively applies the NEH construction mechanism, the Bubble-Swap improvement phase, and the Merge-Global-Swap refinement phase. Each phase performs a finite number of evaluations determined solely by the problem size, and no stochastic decisions or adaptive stopping criteria are involved.

Consequently, BNMG always terminates after a finite number of deterministic operations and produces the same schedule for the same problem instance. This deterministic behavior ensures full reproducibility of the obtained solutions without requiring multiple independent runs.

Since BNMG is not based on an iterative optimization process that repeatedly updates a population or a candidate solution until convergence, the classical concepts of local or global convergence are not directly applicable. Instead, the algorithm provides a deterministic mapping from an input instance to a feasible schedule through a finite sequence of constructive and improvement operations. Accordingly, the effectiveness of BNMG should be assessed by the quality of the generated schedules and computational efficiency rather than by convergence properties that are specific to iterative optimization algorithms.

4.5. Time Complexity Analysis

The proposed BNMG and BNMG-II algorithms are hybrid heuristic methods consisting of multiple stages. Therefore, the total running time of the algorithms is obtained by examining the time complexity of each component separately and then combining them. In this section, the asymptotic running times of the BNMG and BNMG-II algorithms are derived by analyzing the complexity of all sub-procedures used.

Unlike population-based or iterative metaheuristics, BNMG is a deterministic constructive heuristic that executes a single finite sequence of operations. Therefore, the algorithm does not involve repeated iterations until a stopping criterion is met. Consequently, the overall computational complexity is identical to the complexity of this single execution, and no distinction between per-iteration complexity and total complexity is required.

The NEH_order procedure calculates the total processing time of each job and sorts the jobs in descending order according to these values. Calculating the total processing times takes $O(nm)$ time, while sorting the resulting n values takes $O(n \log n)$ time. Therefore, the total complexity of the NEH_order stage can be expressed as $O(nm + n \log n)$ time. In practice, since m is usually greater than $\log n$, this stage often becomes dominant as $O(nm)$.

The Bubble-Swap operator performs approximately $\frac{n(n-1)}{2} = O(n^2)$ neighboring swap attempts using two nested loops. The completion time for each swap evaluation needs to be calculated. Using a full calculation, this takes $O(nm)$ time. In this case, the worst-case complexity of the Bubble-Swap operator is $O(n^3m)$. However, using incremental completion time calculation techniques, each swap evaluation can take $O(m)$ time, and the total complexity can be reduced to $O(n^2m)$ level.

The NEH_construct procedure uses a sorted list of jobs to place them sequentially into partial permutations, testing all possible positions for each placement. While the k th job is placed, k different positions are tried, and the completion time is calculated for each trial. Therefore, the classical complexity of the NEH_construct phase is known as $O(n^2m)$, and the same calculation applies in this study.

The merge-based operator recursively splits the array into two and evaluates potential swaps in each merge step. The splitting process requires $O(n \log n)$ merge steps, similar to the classical merge sort algorithm. However, since the completion time is calculated during each comparison, the cost of each merge step depends on the problem size. When using full computation, the worst-case complexity of this stage can reach the order of $O(n^2m)$. In practice, this value is lower with incremental computation techniques, but the asymptotic upper bound is thus preserved.

The iterative local search procedure used in the final stage of the BNMG-II algorithm tests whether each job is moved to all positions within the permutation. This process requires approximately $O(n^2)$ movement attempts in one iteration. Since the completion time needs to be calculated for each move, the cost of one iteration is $O(n^2m)$. Assuming that I iterations occur, since the local search process continues until no improvement remains, the total complexity of this stage can be expressed as $O(In^2m)$.

When the above analyses are combined, the total time complexity of the BNMG algorithm can be written as $O(nm) + O(n^2m) + O(n^2m) + O(n^2m)$. Considering the dominant terms, the general upper bound for the BNMG algorithm is $O(n^2m)$ (assuming incremental completion-time calculation is used).

For the BNMG-II algorithm, since there is an additional iterative local search phase, the total complexity is $O(In^2m)$. Here, I denotes the number of iterations at which improvement is achieved; in practice, this value is usually small.

Both versions of the algorithms essentially store a work permutation and intermediate tables used to calculate completion times. Therefore, the memory complexity is $O(nm)$. This value is on the same level as other constructive and local search-based heuristic methods in the literature for flow-type scheduling problems.

In conclusion, the proposed BNMG algorithm remains in the same asymptotic complexity class as NEH, but aims to produce higher-quality solutions thanks to additional local and global optimization mechanisms. The BNMG-II algorithm, while having a higher computational cost due to the additional local search phase, justifies this additional cost, especially for large-scale problems, thanks to the improved solution quality.

5. Experimental Design and Computational Results

To evaluate the effectiveness of the proposed heuristic algorithms, a comprehensive set of computational experiments was conducted on test examples of Taillard's permutational flow-type shop scheduling problem (PFSSP) [50], a widely used problem in the literature. The experimental evaluation includes all 120 classical Taillard benchmark instances, covering a wide range of problem dimensions from small to very large scheduling instances. These benchmark sets have been extensively adopted in the PFSSP literature for evaluating deterministic and metaheuristic algorithms and constitute one of the most widely accepted test suites for this problem. The problem examples used were obtained from the online resource at [51]. The best-known solutions and lower and upper bounds reported in the literature for these examples were obtained from the [52]. All algorithms were implemented using the Python 3.11 programming language, and the experiments were conducted on a computer with an Intel Core i7 processor and approximately 16 GB of RAM.

For each benchmark instance, the Relative Percentage Deviation (RPD) is computed as

$$RPD_i = \frac{C_i - C_i^*}{C_i^*},$$

where C_i denotes the makespan produced by the algorithm and C_i^* is the best-known solution for instance i .

The M1 metric reported throughout this study is the Average Relative Percentage Deviation (ARPD), computed over all benchmark instances,

$$M1 = \frac{1}{N} \sum_{i=1}^N RPD_i,$$

where N is the number of benchmark instances. Lower M1 values indicate better solution quality.

To jointly assess solution quality and computational efficiency, we additionally report the M2 metric, defined as

$$M2 = \frac{1}{N} \sum_{i=1}^N \left(0.8RPD_i + 0.2 \frac{t_i}{t_i^{best}} \right),$$

where t_i is the execution time of the evaluated algorithm on instance i , and t_i^{best} denotes the shortest execution time among all algorithms that achieved the best makespan for that instance.

The weighting factors (0.8 and 0.2) assign greater importance to solution quality while still reflecting computational efficiency. Since M2 combines two performance criteria, it is used only as a complementary performance indicator, whereas the primary conclusions of this study are based on direct makespan comparisons and the M1 metric.

The primary performance criterion adopted in this study is the M1 metric, which directly measures the relative percentage deviation from the best known solution. Since solution quality is the primary objective of the PFSSP, all algorithmic comparisons and conclusions are mainly based on M1.

The M2 metric is reported only as a supplementary indicator to provide additional insight into the trade-off between solution quality and computational efficiency. Therefore, M2 should not be interpreted as a replacement for M1, but rather as an auxiliary measure for comparing algorithms with similar solution quality.

Different weight combinations could also be adopted depending on the application requirements. Therefore, the proposed M2 metric is not intended as a universal benchmark, but rather as a complementary evaluation criterion for this study.

Experimental results are given in Tables 1 and 2. These two tables present a comparison of the proposed BNMG and BNMG-II algorithms with the widely used NEH, NEH-II, vN-NEH, and vN-NEH+ algorithms in the literature in terms of solution quality (M1: ARPD metric) and computation time sensitivity (M2: quality + time combined metric). Table 1 shows the average performance values obtained for the first 55 Taillard Problems, and Table 2 shows the average performance values obtained for the first 100 Taillard Problems, which is a larger test set. We give the dependence of the M1 metric on the number of problems in Figure 1.

Table 1. Performance comparison for the first 55 Taillard Problems.

Algorithm	M1	M2
Proposed BNMG	1.49	4.39
NEH	1.74	3.55
vN-NEH	1.83	11.88
Proposed BNMG-II	1.02	14.20
NEH-II	1.06	16.10
vN-NEH+	1.30	71.67

Table 2. Performance comparison for the first 100 Taillard Problems.

Algorithm	M1	M2
Proposed BNMG	2.41	6.81
NEH	2.69	5.60
vN-NEH	2.86	35.19
Proposed BNMG-II	1.72	20.11
NEH-II	1.75	23.91
vN-NEH+	1.95	556.85

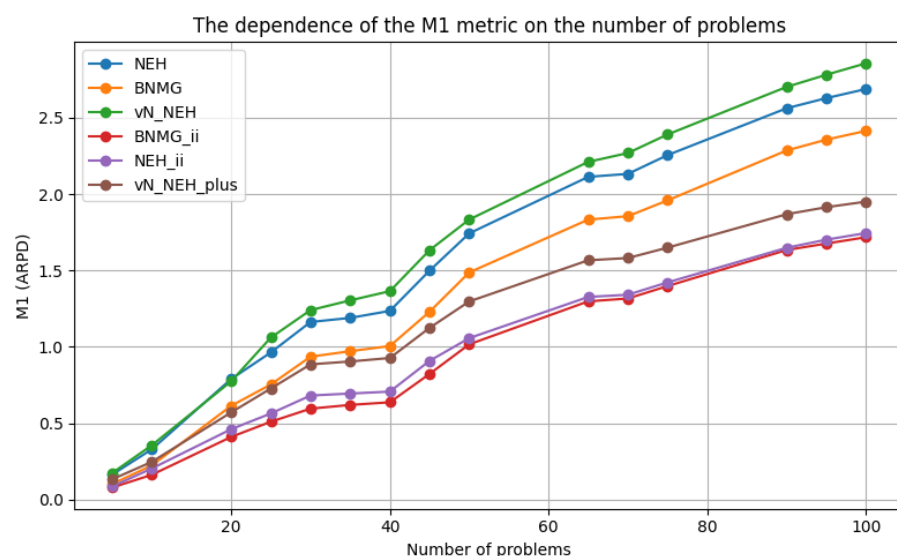


Figure 1. The dependence of the M1 metric on the number of problems.

In the experiment conducted on the first 55 Taillard Problems, the proposed BNMG produced significantly better results than the classical NEH (1.74) and vN-NEH (1.83) algorithms in terms of the M1 metric (Solution Quality), with an M1 value of 1.49. Among

the algorithms with improvement (II), the proposed BNMG-II (1.02) has the lowest M1 value. This value is better than the NEH-II (1.06) and vN-NEH+ (1.30) results. This demonstrates that the proposed algorithm consistently produces higher-quality schedules than the compared deterministic heuristics while maintaining the computational efficiency of deterministic search methods. In terms of M2 (Quality + Time), BNMG, with M2 = 4.39, is slightly slower than NEH (3.55), but exhibits a much more balanced and efficient performance compared to vN-NEH (11.88). Among the algorithms with improvements, BNMG-II (14.20) is better than NEH-II (16.10), and significantly superior to vN-NEH+ (71.67). These results clearly show that vN-NEH+ is inefficient in practice due to its excessively high computation time, even though its solution quality is reasonable.

Table 2 shows the experiments conducted on the first 100 Taillard Problems, the trends become even more pronounced in the larger problem set. In terms of M1, BNMG (2.41) performed better than both NEH (2.69) and vN-NEH (2.86). Among the algorithms with improvements, BNMG-II (1.72) gave the best result, while NEH-II (1.75) was a very close second. vN-NEH+ (1.95) lagged behind both algorithms. This table shows that the scalability of the proposed algorithm is strong as the problem size increases. In terms of M2, an extremely high value of M2 = 556.85 was obtained for vN-NEH+. This shows that the algorithm's computation time completely overshadows the quality gain. In contrast, BNMG-II (20.11) offers a better balance than NEH-II (23.91). In particular, this result clearly shows that the vN-NEH+ algorithm, published in 2024, is weak in terms of time/efficiency, even compared to the classical NEH-II with improvements.

Table 3 presents the results obtained for the last 20 Taillard benchmark instances, which correspond to the largest and most computationally demanding problems in the benchmark set. These instances provide a more challenging environment for evaluating the scalability and robustness of deterministic constructive heuristics.

Table 3. Performance comparison for the last 20 Taillard Problems.

Algorithm	M1	M2
Proposed BNMG	0.58	0.62
NEH	0.62	0.64
vN NEH	0.50	8.26
Proposed BNMG-II	0.46	3.24
NEH-II	0.48	3.38

The results indicate that the proposed BNMG algorithm achieves an average M1 value of 0.58, outperforming the classical NEH algorithm (0.62), while the proposed BNMG-II further improves the solution quality with an M1 value of 0.46. Although the vN-NEH algorithm achieves a slightly lower M1 value (0.50) than BNMG, this improvement is accompanied by a substantially larger M2 value (8.26), indicating considerably higher computational cost. Therefore, BNMG provides a more balanced trade-off between solution quality and computational efficiency.

Similarly, BNMG-II achieves the best compromise among the improvement-based heuristics by obtaining both lower M1 and M2 values than NEH-II. These results suggest that the proposed Merge-Global-Swap mechanism becomes particularly effective when combined with the iterative improvement phase, allowing better schedules to be produced without excessive computational overhead.

Figure 1 illustrates how the average M1 value changes as the number of benchmark instances increases. It can be observed that the relative ranking of the algorithms remains stable throughout the benchmark set. In particular, BNMG consistently performs better than the classical NEH and remains competitive with recently proposed deterministic

heuristics. Moreover, BNMG-II maintains the lowest average M1 values over almost the entire evaluation range, indicating that the proposed hybridization strategy preserves its effectiveness as the benchmark size grows. This behavior suggests that the proposed global decision mechanism contributes not only to improving solution quality on individual instances but also to maintaining stable performance across problems of different sizes.

This observation indicates that the proposed BNMG algorithm prioritizes a balanced optimization strategy rather than pursuing marginal improvements in solution quality at the expense of computational efficiency.

When these two tables are evaluated together, the following results are clearly evident:

- BNMG produces better solution quality compared to NEH and vN-NEH while maintaining its deterministic structure.
- BNMG-II achieves the best solution quality (M1) among the proposed improvement-based deterministic heuristics while maintaining competitive computational performance. Although the classical NEH algorithm exhibits lower M2 values due to its simpler structure and lower computational cost, BNMG-II provides a substantially better balance between solution quality and computational efficiency through its improved scheduling performance.
- While vN-NEH and especially vN-NEH+ appear competitive in terms of solution quality, they lose practical applicability when computation time is considered.
- Thanks to the proposed M2 metric, deterministic algorithms are objectively evaluated not only for “how well they find a solution” but also for how long it takes them to reach that solution. Although M2 combines solution quality and runtime into a single indicator, the optimization objective of the PFSSP remains minimizing the makespan. Therefore, M1 is considered the principal performance metric throughout this study, whereas M2 is presented only to provide complementary information regarding computational efficiency. Although M2 provides a convenient way of jointly considering solution quality and runtime, its weighting coefficients are predefined and therefore may not reflect every practical scenario. For this reason, M2 is used only as a complementary performance indicator in this study, while all primary comparisons rely on the widely accepted M1 (ARPD) metric.

Table 3 summarizes the results obtained for the last 20 Taillard benchmark instances, which represent the largest and most computationally demanding problems in the benchmark suite. These large-scale instances provide a rigorous evaluation of the scalability and robustness of deterministic heuristic algorithms.

Although vN-NEH achieves a slightly lower average M1 value (0.50) than the proposed BNMG algorithm (0.58), the observed difference is relatively small. This indicates that the insertion mechanism employed by vN-NEH remains highly competitive for these particularly challenging instances, leaving limited room for further improvement during the constructive phase. Nevertheless, the proposed BNMG algorithm still outperforms the classical NEH heuristic in both M1 and M2, demonstrating that the Bubble-Swap and Merge-Global-Swap mechanisms consistently enhance the solution quality without increasing the computational cost.

The effectiveness of the proposed framework becomes more evident when the improvement phase is incorporated. As shown in Table 3, BNMG-II achieves the best average M1 value (0.46), outperforming both vN-NEH (0.50) and NEH-II (0.48). Furthermore, BNMG-II also attains a lower M2 value than NEH-II, indicating a better balance between solution quality and computational efficiency among the improvement-based deterministic heuristics.

Since the proposed algorithms are deterministic, identical input instances always produce identical schedules. Therefore, their stability is reflected by the consistency of their performance across different benchmark groups rather than by repeated-run variability.

The experimental results demonstrate that BNMG and BNMG-II maintain competitive performance over small, medium, and large Taillard benchmark instances, confirming the robustness of the proposed framework.

5.1. Ablation Study

To evaluate the contribution of each component of the proposed BNMG algorithm, an ablation study was conducted on the first 100 Taillard benchmark instances. Three reduced variants were considered by removing one module at a time while keeping the remaining algorithm unchanged.

The results are reported in Table 4. Removing the Bubble-Swap stage slightly degrades the average M1 value from 2.41 to 2.49, indicating that this stage contributes to improving the initial sequence before the constructive phase. Eliminating the Merge-Global-Swap stage has a larger impact, increasing the M1 value to 2.66, which confirms the effectiveness of the global refinement process.

Table 4. Ablation study of the proposed BNMG algorithm on the first 100 Taillard benchmark instances.

Configuration	M1
Proposed BNMG	2.41
Without Bubble-Swap	2.49
Without NEH-like Construction	11.78
Without Merge-Global-Swap	2.66

The most significant deterioration is observed when the NEH-like construction stage is removed. In this case, the average M1 value increases dramatically to 11.78, demonstrating that this component forms the structural foundation of the proposed heuristic. The Bubble-Swap and Merge-Global-Swap operators therefore act as complementary improvement mechanisms built upon this construction phase.

Overall, the ablation results demonstrate that each module contributes positively to the final performance of BNMG, while the combination of all three components yields the best overall solution quality.

5.2. Runtime Analysis

Table 5 presents the runtime statistics of all deterministic algorithms over the 120 Taillard benchmark instances. As expected, the classical NEH heuristic is the fastest method, with an average runtime of 9.665 s. The proposed BNMG algorithm requires a very similar computational effort, with an average runtime of 9.988 s, corresponding to only a 3.3% increase. Considering the consistent improvements in solution quality reported in Tables 1–3, this additional computational cost is negligible in practice.

Table 5. Runtime statistics of the deterministic algorithms over all 120 Taillard benchmark instances.

Algorithm	Mean (s)	Std. Dev.	Median (s)	Total (s)	Relative Speed
Proposed BNMG	9.988	29.543	0.363	1198.592	1.00×
NEH	9.665	29.048	0.363	1159.768	0.97×
vN-NEH	686.090	2221.055	3.966	82,330.788	68.69×
Proposed BNMG-II	193.070	650.792	2.130	23,168.411	19.33×
NEH-II	209.797	696.491	2.376	25,175.651	21.00×
vN-NEH+	360.586	890.922	6.903	43,270.266	36.10×

In contrast, the optimization-based variants require substantially longer execution times. BNMG-II and NEH-II increase the average runtime by factors of approximately 19.3 and 21.0,

respectively, while vN-NEH+ and vN-NEH are approximately 36.1 and 68.7 times slower than BNMG. Similar observations can also be made from the median runtimes, where BNMG (0.363 s) remains comparable to NEH (0.363 s), whereas the remaining methods require considerably longer execution times.

These results indicate that the proposed BNMG algorithm successfully improves schedule quality while preserving the computational efficiency of the original NEH heuristic. Therefore, unlike optimization-intensive variants that obtain better solutions at the expense of significantly increased runtime, BNMG achieves a favorable balance between solution quality and computational cost, making it particularly suitable for large-scale scheduling problems and time-sensitive production environments.

The experimental runtime analysis also supports the theoretical complexity discussion presented in Section 4.5. Although BNMG introduces additional Bubble-Swap and Merge-Global optimization stages, these operations incur only a marginal computational overhead compared with the original NEH heuristic. Consequently, the practical runtime remains almost unchanged while providing consistently improved scheduling performance.

5.3. Statistical Significance Analysis

Since all compared algorithms are deterministic, repeated executions on the same benchmark instance always produce identical schedules. Therefore, reporting standard deviations over multiple runs would not provide additional information. Instead, statistical significance was evaluated by considering the makespan obtained for each Taillard benchmark instance as paired observations.

The proposed algorithms were compared with the competing deterministic heuristics using the Wilcoxon signed-rank test. Since multiple pairwise comparisons were performed, Holm–Bonferroni correction was applied to control the family-wise error rate. In addition to statistical significance, the practical importance of the observed differences was quantified using Cliff’s Delta (δ) and the Vargha–Delaney effect size (A_{12}). Finally, Win/Tie/Loss statistics were reported to evaluate the consistency of the proposed algorithms over the complete benchmark suite.

Table 6 presents the statistical comparison over all 120 Taillard benchmark instances. The results indicate that BNMG achieves a statistically significant improvement over the classical NEH heuristic ($p = 0.0153$), although the corresponding effect size is small to medium ($\delta = 0.208$). In contrast, the differences between BNMG and vN-NEH, as well as between BNMG-II and NEH-II, are not statistically significant over the complete benchmark suite. This observation suggests that these algorithms exhibit comparable overall performance despite differences in average M_1 and M_2 values.

Table 6. Statistical comparison over all 120 Taillard benchmark instances.

Comparison	Wilcoxon p	Cliff’s δ	A_{12}	Win/Tie/Loss
BNMG vs. NEH	0.0153	0.208	0.604	69/7/44
BNMG vs. vN-NEH	0.6050	0.117	0.558	66/2/52
BNMG-II vs. NEH-II	0.7370	0.075	0.538	57/15/48
BNMG-II vs. BNMG	2.60×10^{-17}	0.792	0.896	95/25/0

The comparison between BNMG-II and BNMG reveals a highly significant improvement ($p < 10^{-16}$), accompanied by a large effect size ($\delta = 0.792$, $A_{12} = 0.896$). Furthermore, BNMG-II achieves 95 wins, 25 ties, and no losses against BNMG, demonstrating that the second-stage improvement consistently enhances solution quality across a wide range of benchmark instances.

6. Conclusions

In this study, a new deterministic hybrid algorithm called BNMG (Bubble-NEH-Merge-Global) is proposed for the Permutation Flow-Type Workshop Scheduling Problem (PFSSP). Unlike the widely used NEH and its derivatives (NEH-II, vN-NEH, and vN-NEH+) in the literature, the proposed approach does not rely solely on local or partial evaluations. The method developed in this study offers a global evaluation approach that performs decision processes based on the makespan value of the entire job sequence, particularly through the Merge-Global-Swap component.

In existing deterministic methods in the literature, job addition or replacement decisions are mostly evaluated based on partial sequences or limited neighborhood structures, which can lead to irreversible sub-optimal decisions made in the early stages, especially in large-scale problems. In contrast, the global makespan-based swap criterion proposed in this study ensures that each swap decision is directly compatible with the objective function.

The proposed BNMG algorithm combines local optimization with the Bubble-Swap operator, constructive solution generation with an NEH-style insertion mechanism, and global optimization with the Merge-Global-Swap approach under one roof. This integrated structure extends the classical local decision paradigm of deterministic algorithms, allowing for a more efficient exploration of the solution space. Furthermore, the M2 performance metric proposed in this study contributes to a more comprehensive analysis of deterministic algorithms by evaluating solution quality and computation time together.

Experimental studies show that the proposed BNMG and BNMG-II algorithms exhibit superior performance compared to existing deterministic methods in the literature in terms of both solution quality and computational efficiency. The advantages provided by the global decision mechanism are particularly evident in medium- and large-scale problem examples.

From a practical perspective, the proposed BNMG and BNMG-II algorithms are particularly suitable for manufacturing environments where production schedules must be generated rapidly while maintaining high solution quality. Since both algorithms are deterministic, they always produce the same schedule for the same input, making them attractive for industrial decision-support systems that require repeatable and predictable scheduling decisions. Potential application areas include automotive assembly lines, electronics manufacturing, textile production, metal processing, and other flow-shop production systems where makespan minimization directly affects productivity and resource utilization.

Future research may extend the proposed framework to more complex scheduling environments, including no-wait, no-idle, blocking, distributed, and flexible flow shop scheduling problems. Another promising direction is to integrate the proposed deterministic construction strategy into population-based metaheuristics as an initialization mechanism or as a local improvement operator. In addition, parallel implementations and multi-objective extensions could further improve the applicability of the proposed approach to large-scale industrial scheduling problems.

In conclusion, the hybrid deterministic approach proposed in this study offers an effective and reliable solution method for PFSSP, both theoretically and practically, and makes significant contributions to the development of deterministic scheduling algorithms.

Author Contributions: Conceptualization, Y.A. and N.K.; methodology, Y.A.; software, N.K.; formal analysis, N.K.; investigation, Y.A.; data curation, N.K.; writing—original draft preparation, N.K.; writing—review and editing, Y.A. and N.K.; visualization, N.K.; supervision, Y.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: This study used public benchmark data. Computer code can be easily generated from the pseudocode presented in the study, but it may be made available upon request.

Acknowledgments: The authors sincerely thank the Editor and the anonymous Reviewers for their careful evaluation of this manuscript and for their thoughtful and constructive comments. Their suggestions have helped us improve the manuscript in many aspects, including the presentation of the proposed method, the experimental evaluation, and the discussion of the results. We are grateful for the opportunity to revise our work and believe that the manuscript has been significantly strengthened through this review process. No Generative Artificial Intelligence (GenAI) tools or Large Language Models (LLM) were used to generate the scientific content, research ideas, data, figures, study designs, experimental results, analyses, interpretations, or conclusions in our article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Hazrati Nejad, E.; Yigit-Sert, S.; Emrah Amrahov, Ş. An effective global path planning algorithm with teaching-learning-based optimization. *Kybernetika* **2024**, *60*, 293–316. [\[CrossRef\]](#)
2. Yıldırım, H.B.; Kullu, K.; Emrah Amrahov, Ş. A graph model and a three-stage algorithm to aid the physically disabled with navigation. *Univers. Access Inf. Soc.* **2024**, *23*, 901–911.
3. Ar, Y.; Bostanci, E. A genetic algorithm solution to the collaborative filtering problem. *Expert Syst. Appl.* **2016**, *61*, 122–128. [\[CrossRef\]](#)
4. Sert, S.Y.; Ar, Y.; Bostanci, G.E. Evolutionary approaches for weight optimization in collaborative filtering-based recommender systems. *Turk. J. Electr. Eng. Comput. Sci.* **2019**, *27*, 2121–2136. [\[CrossRef\]](#)
5. Kartli, N. Hybrid algorithms for fixed charge transportation problem. *Kybernetika* **2025**, *61*, 141–167. [\[CrossRef\]](#)
6. Kartli, N.; Bostanci, E.; Guzel, M.S. A new algorithm for optimal solution of fixed charge transportation problem. *Kybernetika* **2023**, *59*, 45–63. [\[CrossRef\]](#)
7. Kartli, N.; Bostanci, E.; Guzel, M.S. Heuristic algorithm for an optimal solution of fully fuzzy transportation problem. *Computing* **2024**, *106*, 3195–3227. [\[CrossRef\]](#)
8. Ghareeb, A.; Nooruldeen, O.; Arslan, C.A.; Kapp, S.; Choi, J.K. Synergistic optimization of predictive models for water quality analysis in treatment plants using machine learning and evolutionary algorithms. *Evol. Intell.* **2025**, *18*, 34. [\[CrossRef\]](#)
9. Cetin, P.; Amrahov, Ş.E. A new network-based community detection algorithm for disjoint communities. *Turk. J. Electr. Eng. Comput. Sci.* **2022**, *30*, 2190–2205. [\[CrossRef\]](#)
10. Cetin, P.; Emrah Amrahov, S. A new overlapping community detection algorithm based on similarity of neighbors in complex networks. *Kybernetika* **2022**, *58*, 277–300. [\[CrossRef\]](#)
11. Kartli, N.; Cetin, P.; Ayhan, S. New gravitational algorithms for the detection of overlapping and disjoint communities in weighted complex networks. *Kybernetika* **2025**, *61*, 509–536. [\[CrossRef\]](#)
12. Shao, C.; Yu, Z.; Ding, H.; Cao, G.; Zhou, B. A reinforcement learning method for flexible job shop scheduling based on multi-head attention and deep residual network. *Comput. Electr. Eng.* **2025**, *123*, 110044. [\[CrossRef\]](#)
13. Zhang, J.; Lü, Z.; Ding, J.; Su, Z.; Li, X.; Gao, L. An effective local search algorithm for flexible job shop scheduling in intelligent manufacturing systems. *Engineering* **2025**, *50*, 117–127. [\[CrossRef\]](#)
14. Geetha, M.; Chandra Guru Sekar, R.; Marichelvam, M. A hybrid honey badger algorithm to solve energy-efficient hybrid flow shop scheduling problems. *Processes* **2025**, *13*, 174. [\[CrossRef\]](#)
15. Cubukcuoglu, A.; Karacan, I.; Ceylan, Z.; Bulkan, S. Minimizing Makespan in Ordered Flow Shop Scheduling Using a Robust Genetic Algorithm. *Processes* **2025**, *13*, 1583. [\[CrossRef\]](#)
16. Sun, Z.W.; Lv, D.Y.; Wei, C.M.; Wang, J.B. Flow shop scheduling with shortening jobs for makespan minimization. *Mathematics* **2025**, *13*, 363. [\[CrossRef\]](#)
17. Yin, K.; Li, Z.; Li, M.; Xue, Y.; Chen, Y. A Novel Imperialist Competitive Algorithm for Energy-Efficient Permutation Flow Shop Scheduling Problem Considering the Deterioration Effect of Machines. *Mathematics* **2025**, *13*, 3973. [\[CrossRef\]](#)
18. Nawaz, M.; Ensco, E.E., Jr.; Ham, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **1983**, *11*, 91–95. [\[CrossRef\]](#)
19. Puka, R.; Skalna, I.; Duda, J.; Stawowy, A. Deterministic constructive vN-NEH+ algorithm to solve permutation flow shop scheduling problem with makespan criterion. *Comput. Oper. Res.* **2024**, *162*, 106473. [\[CrossRef\]](#)
20. Fernandez-Viagas, V.; Framinan, J.M. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Comput. Oper. Res.* **2014**, *45*, 60–67. [\[CrossRef\]](#)

21. Rajendran, C. Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *Int. J. Prod. Econ.* **1993**, *29*, 65–73. [\[CrossRef\]](#)
22. Dong, X.; Huang, H.; Chen, P. An improved NEH-based heuristic for the permutation flowshop problem. *Comput. Oper. Res.* **2008**, *35*, 3962–3968. [\[CrossRef\]](#)
23. Kalczynski, P.J.; Kamburowski, J. An improved NEH heuristic to minimize makespan in permutation flow shops. *Comput. Oper. Res.* **2008**, *35*, 3001–3008. [\[CrossRef\]](#)
24. Puka, R.; Duda, J.; Stawowy, A.; Skalna, I. N-NEH+ algorithm for solving permutation flow shop problems. *Comput. Oper. Res.* **2021**, *132*, 105296. [\[CrossRef\]](#)
25. Pan, Q.K.; Ruiz, R. Local search methods for the flowshop scheduling problem with flowtime minimization. *Eur. J. Oper. Res.* **2012**, *222*, 31–43. [\[CrossRef\]](#)
26. Pan, Q.K.; Ruiz, R. An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. *Omega* **2012**, *40*, 166–180. [\[CrossRef\]](#)
27. Tasgetiren, M.F.; Pan, Q.K.; Kizilay, D.; Gao, K. A variable block insertion heuristic for the blocking flowshop scheduling problem with total flowtime criterion. *Algorithms* **2016**, *9*, 71. [\[CrossRef\]](#)
28. Amrahov, S.E.; Alsalihe, T.A.I. Greedy algorithm for the scheduling aircrafts landings. In *Proceedings of the 2011 5th International Conference on Application of Information and Communication Technologies (AICT)*; IEEE: New York, NY, USA, 2011; pp. 1–3.
29. Zhao, Z.; Zhou, M.; Liu, S. Iterated greedy algorithms for flow-shop scheduling problems: A tutorial. *IEEE Trans. Autom. Sci. Eng.* **2021**, *19*, 1941–1959. [\[CrossRef\]](#)
30. Zhang, X.; Zhan, Z.H.; Zhang, J. A fast efficient local search-based algorithm for multi-objective supply chain configuration problem. *IEEE Access* **2020**, *8*, 62924–62931. [\[CrossRef\]](#)
31. Zhao, C.; Wu, L.-H.; Zuo, C.-L.; Zhang, H.-Q.; Xiao, Q. An adaptive dynamic neighborhood crow search algorithm for solving permutation flow shop scheduling problems. *J. Ind. Manag. Optim.* **2024**, *20*, 84–111. [\[CrossRef\]](#)
32. Framinan, J.M.; Leisten, R.; Ruiz-Usano, R. Comparison of heuristics for flowtime minimisation in permutation flowshops. *Comput. Oper. Res.* **2005**, *32*, 1237–1254. [\[CrossRef\]](#)
33. França, P.M.; Gendreau, M.; Laporte, G.; Müller, F.M. A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. *Comput. Oper. Res.* **1994**, *21*, 205–210. [\[CrossRef\]](#)
34. Ruiz, R.; Maroto, C. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *Eur. J. Oper. Res.* **2006**, *169*, 781–800. [\[CrossRef\]](#)
35. Laha, D.; Chakraborty, U.K. A constructive heuristic for minimizing makespan in no-wait flow shop scheduling. *Int. J. Adv. Manuf. Technol.* **2009**, *41*, 97–109.
36. Reeves, C.R. A genetic algorithm for flowshop sequencing. *Comput. Oper. Res.* **1995**, *22*, 5–13. [\[CrossRef\]](#)
37. Ribas, I.; Companys, R.; Tort-Martorell, X. Efficient heuristics for the parallel blocking flow shop scheduling problem. *Expert Syst. Appl.* **2017**, *74*, 41–54. [\[CrossRef\]](#)
38. Ruiz, R.; Pan, Q.K.; Naderi, B. Iterated Greedy methods for the distributed permutation flowshop scheduling problem. *Omega* **2019**, *83*, 213–222. [\[CrossRef\]](#)
39. Chen, S.; Zheng, J. Hybrid grey wolf optimizer for solving permutation flow shop scheduling problem. *Concurr. Comput. Pract. Exp.* **2024**, *36*, e7942.
40. Fazekas, L.; Tüü-Szabó, B.; Kóczy, L.T.; Hornyák, O.; Nehéz, K. A Hybrid Discrete Memetic Algorithm for Solving Flow-Shop Scheduling Problems. *Algorithms* **2023**, *16*, 406. [\[CrossRef\]](#)
41. Hayat, I.; Tariq, A.; Shahzad, W.; Masud, M.; Ahmed, S.; Ali, M.U.; Zafar, A. Hybridization of particle swarm optimization with variable neighborhood search and simulated annealing for improved handling of the permutation flow-shop scheduling problem. *Systems* **2023**, *11*, 221. [\[CrossRef\]](#)
42. Khurshid, B.; Maqsood, S.; Omair, M.; Sarkar, B.; Saad, M.; Asad, U. Fast evolutionary algorithm for flow shop scheduling problems. *IEEE Access* **2021**, *9*, 44825–44839. [\[CrossRef\]](#)
43. Mishra, A.; Shrivastava, D. A discrete Jaya algorithm for permutation flow-shop scheduling problem. *Int. J. Ind. Eng. Comput* **2020**, *11*, 415–428. [\[CrossRef\]](#)
44. Mishra, A.; Shrivastava, D. Parameter less hybrid IG-Jaya approach for permutation flow shop scheduling problem. *J. Ind. Manag. Optim.* **2024**, *20*, 606–629. [\[CrossRef\]](#)
45. de Fátima Morais, M.; Ribeiro, M.H.D.M.; da Silva, R.G.; Mariani, V.C.; dos Santos Coelho, L. Discrete differential evolution metaheuristics for permutation flow shop scheduling problems. *Comput. Ind. Eng.* **2022**, *166*, 107956. [\[CrossRef\]](#)
46. Dodu, C.E.; Ancau, M. An efficient algorithm for lot permutation flow shop scheduling problem. *Proc. Rom. Acad. Ser. A* **2021**, *22*, 231–238.
47. Zhang, Z.; Zhao, W.; Zhao, H.; Bian, X. MNATS: A Multi-Neighborhood Adaptive Tabu Search Algorithm for the Distributed No-Wait Flow Shop Scheduling Problem. *Appl. Sci.* **2025**, *15*, 9840. [\[CrossRef\]](#)
48. Koulamas, C. A new constructive heuristic for the flowshop scheduling problem. *Eur. J. Oper. Res.* **1998**, *105*, 66–71. [\[CrossRef\]](#)

49. Framinan, J.M.; Gupta, J.N.; Leisten, R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *J. Oper. Res. Soc.* **2004**, *55*, 1243–1255. [[CrossRef](#)]
50. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [[CrossRef](#)]
51. Chneau. Go-Taillard: Flowshop Scheduling Problem (Taillard Instances) in Go. 2025. Available online: <https://github.com/chneau/go-taillard/tree/master/pfsp/instances> (accessed on 29 January 2026).
52. Brunner, O. New Best Results for Some Known Instances of the Permutation FlowShop Problem (PFSP) (Version v10e). 2026. Available online: <https://zenodo.org/records/20612410> (accessed on 29 January 2026). [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.