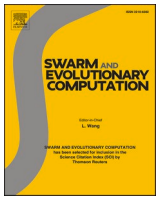
Swarm and Evolutionary Computation 97 (2025) 102028



Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo



Greedy randomized adaptive search and benders decomposition algorithms to solve the distributed no-idle permutation flowshop scheduling problem

Alper Hamzadayı [a,*], Münevver Günay Van [b]

[a] Department of Industrial Engineering, Van Yuzuncu Yil University, 65080 Van, Türkiye
[b] The Graduate School of Natural and Applied Sciences, Van Yuzuncu Yil University, 65080 Van, Türkiye

ARTICLE INFO

Keywords:
Distributed no-idle flowshop problem
Greedy randomized adaptive search
Benders decomposition algorithm
Mathematical models
LS3 algorithm

ABSTRACT

In today's competitive manufacturing landscape, large enterprises manage multiple production sites, leading to complex scheduling challenges. This study investigates the Distributed No-Idle Permutation Flowshop Scheduling Problem (DNIPFSP), where the objective is to minimize makespan across multiple identical factories while ensuring continuous machine utilization without idle time. To address this problem, we propose both approximation and exact methods. For the approximation method, we introduce a novel Greedy Randomized Adaptive Search Procedure (GRASP). On the exact optimization side, we develop three mathematical formulations: a sequence-based model, an improved position-based model, and a restricted version of the improved position-based model, where the upper bounds of decision variables are determined through a two-stage process. First, an initial GRASP solution is obtained, and based on this solution, an additional model is solved to compute the upper bounds of decision variables. The Benders decomposition algorithm is then applied to efficiently solve problem instances. To further improve computational efficiency, we introduce a hybrid Benders decomposition algorithm, incorporating heuristic-derived cuts alongside standard Benders cuts. Additionally, symmetry-breaking constraints are integrated to strengthen the formulations. Extensive benchmark experiments demonstrate the superiority of the proposed methods over existing approaches. The hybrid Benders decomposition algorithm with symmetry-breaking constraints significantly outperforms the best-known models in the literature, optimally solving 419 out of 420 small-sized instances with an average optimality gap of 0.011%. Additionally, the GRASP achieves the lowest average relative percentage deviation (RPD) for large-sized instances, demonstrating its effectiveness in large-scale scheduling optimization.

1. Introduction

In today's globalized and decentralized economy, the importance of multi-factory systems is on the rise [1–4]. On the other hand, researchers have concentrated on narrowing the gap between theoretical and practical scheduling by examining more practical adaptations of the flowshop problem to more effectively tackle real-world challenges [5]. As a result, there is growing focus on production scheduling within these multi-factory settings, commonly called the distributed scheduling problem [6–9]. Distributed permutation flowshop scheduling (DPFSP) is a key challenge in distributed scheduling, and a variation of the standard permutation flowshop scheduling problem (PFSP). The PFSP involves processing N jobs across F identical factories, each containing M machines organized in a flowshop arrangement. The objective is to distribute jobs across factories and define the processing sequence based on certain performance measures. Two primary objectives in scheduling research are minimizing makespan and total flowtime. In DPFSP, minimizing makespan improves production efficiency, reduces costs, and enhances customer satisfaction.

Introduced by Naderi and Ruiz [10], DPFSP focuses on minimizing makespan across multiple factories. They pointed out that this problem presents greater complexity compared to PFSP, which involves job scheduling within a single factory. To tackle this, they introduced two constructive heuristic methods and further refined their results through the application of a Variable Neighborhood Search (VNS) metaheuristic. Since their groundbreaking work, a wide range of heuristic and metaheuristic methods have been proposed to tackle the DPFSP, primarily aimed at reducing the makespan.

Several heuristic and metaheuristic approaches have been proposed to solve the Distributed Permutation Flowshop Scheduling Problem

* Corresponding author.
E-mail addresses: alperhamzadayi@yyu.edu.tr (A. Hamzadayı), m.gunayvan@gmail.com (M.G. Van).

https://doi.org/10.1016/j.swevo.2025.102028
Received 30 December 2024; Received in revised form 15 March 2025; Accepted 30 May 2025
Available online 12 June 2025
2210-6502/© 2025 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

(DPFSP) with the objective of minimizing makespan. One of the earliest contributions was an improved version of the NEH2 heuristic introduced by Naderi and Ruiz [10], incorporating a new insertion rule [11]. Subsequently, various advanced algorithms have been developed, including a hybrid genetic algorithm (HGA) with an enhanced local search technique [12], a modified Variable Neighborhood Descent (VND) that integrates an improved NEH heuristic [13], and a tabu search (TS) algorithm, which explores neighboring solutions by swapping subsequences between factories [14]. Further developments in metaheuristics include a modified iterated greedy (MIG) approach [15], an estimation of distribution algorithm (EDA) that models explicit probability distributions [16], and a scatter search metaheuristic [17]. Other notable techniques involve a hybrid immune algorithm (HIA) [18], a bound-search iterated greedy (BSIG) method combining multiple local search strategies [19], and a chemical reaction optimization algorithm [20]. More recent studies have introduced a two-stage iterated greedy algorithm (IG2S) [21], an artificial bee colony (ABC) algorithm [22], and a multi-agent system based on Nash Q-learning [23]. Additionally, Iterated Greedy (IG) methods have been successfully applied to solve DPFSP with sequence-dependent setup times [24]. Beyond these metaheuristic approaches, Hamzadayı [25] introduced a Benders decomposition-based exact solution method specifically designed to minimize makespan in DPFSP. This demonstrates that, while metaheuristics dominate the literature, exact methods have also been explored to enhance solution quality and guarantee optimality in certain cases.

The no-idle permutation flowshop scheduling problem (NIPFSP) plays a crucial role in manufacturing sectors like fiberglass production and foundries, where machine setup costs are significant, making the stoppage and restarting of machines impractical [26–28]. As a result, many managers choose to extend job completion times by delaying the initiation of the first job on each machine. This challenge was initially explored and proven to be NP-Complete [29], prompting the development of various algorithms aimed at solving NIPFSPs [30–32]. The IG algorithm introduced by Ruiz and Stutzle [33] was modified by Pan and Ruiz [26] to focus on minimizing makespan in the mixed NIPFSP. Their IG_{PR} algorithm integrates aspects from the FRB41 algorithm [34], enhancing the initialization and reconstruction steps using insertion neighborhoods, which boosts diversity in the search for optimal solutions. A numerical comparison of 1750 instances revealed that the IG_{PR} algorithm outperformed other existing heuristics. Furthermore, IG-based algorithms have proven effective in addressing various scheduling problems [26,33,35–39], consistently showcasing superior performance in both tractability and near-optimal solutions for the NIPFSP.

The first contribution to no-idle DPFS problems (DNIPFSPs) is Ying et al. [9], who introduced a position-based mixed integer linear programming (MILP) model along with an Iterated Reference Greedy (IRG) algorithm to address the problem. Additionally, they expanded the test set from Naderi and Ruiz [10] by introducing new instances for evaluating their approach, based on the benchmark by Vallada et al. [40] for the permutation flowshop problem, labeled as TB2. Their method was compared to approach proposed by Pan and Ruiz [26] for NIPFSP, focused on minimizing makespan, but did not incorporate distributed scheduling strategies. In the mixed no-idle permutation flow shop problem (mixed NIPFSP), most machines are permitted to have idle periods, while others must operate continuously without idle periods. Cheng et al. [41] investigated the mixed DNIPFSP with the objective of minimizing the makespan. They developed a MILP model inspired by the works of Naderi and Ruiz [10] and Pan and Ruiz [26], which addressed the DPFS and the mixed no-idle PFSP, respectively, both focusing on minimizing the makespan. Chen et al. [42] investigated the energy-efficient distributed no-idle permutation flow-shop scheduling problem, focusing on reducing both makespan and total energy consumption. They developed a collaborative optimization algorithm (COA) combined with specific collaborative strategies to efficiently solve the problem. Focusing on total completion time as the objective, Li et al. [43] tackled the mixed DNIPFSP by introducing a MILP model and an IG algorithm. To efficiently compute the objective function, they utilized a speed-up procedure.

This paper advances this area of research by tackling DNIPFSP, specifically aiming to minimize the makespan. As mentioned earlier, minimizing makespan and total flowtime are central goals in scheduling research. When focusing on the DNIPFSP and aiming to minimize makespan, the problem can be expressed as $DF_m|prmu;\ no_idle|C_{max}$ [9] following the notation introduced by Graham et al. [44]. Since this problem, when limited to a single factory, reduces to the NP-Complete NIPFSP [29], it can be concluded that $DF_m|prmu;\ no_idle|C_{max}$ problem is also NP-Complete [9]. To the best of our knowledge, the only study addressing $DF_m|prmu;\ no_idle|C_{max}$ is by Ying et al. [9]. Perez-Gonzalez and Framinan [45] provide an extensive review and classification on DPFSPs, offering more in-depth insights for those looking to explore this area further.

To bridge the gap for $DF_m|prmu;\ no_idle|C_{max,}$ both exact and approximation algorithms are included in this paper. Specifically, we introduce three MILP models along with a greedy randomized adaptive search algorithm (GRASP) [46,47] to effectively solve this problem. In this paper, LS3 algorithm is modified so that it can improve not only the critical factory (with the highest makespan) but also other factories, and is integrated into GRASP as a local search. Although GRASP is widely used to solve scheduling problems in the literature, it has, to our knowledge, not yet been applied to any variant of DPFSPs. Two of MILP models mentioned above are the sequence-based (SB) model and the improved position-based (IV_PB) model. This paper also introduces a procedure that combines another MILP model with a heuristic solution obtained through GRASP to determine an upper bound value for the decision variables in IV_PB model, aiming to eliminate unnecessary extra positions in IV_PB model. Subsequently, a new MILP model based on IV_PB model is developed by restricting the decision variables using the obtained upper bound value, referred to as RIV_PB model. Initially, the traditional Benders decomposition algorithm is applied to the RIV_PB model. To speed up convergence, a hybrid method is then generated by combining the RIV_PB model with GRASP within the Benders decomposition structure.

This paper is organized as follows: In Section 2, the PB model by Ying et al. [9] along with the proposed SB and IV_PB models are presented. In Section 3, a procedure is outlined that combines a MILP model with a heuristic approach to calculate an upper bound for the decision variables in the IV_PB model. Section 4 introduces the RIV_PB model, which is derived from the IV_PB model by limiting the number of decision variable positions. Section 5 covers the proposed Benders decomposition algorithms, while Section 6 explains the GRASP algorithm. Section 7 delves into the computational experiments, and Section 8 gives a concluding overview of the paper.

2. Mathematical formulations

This section presents the important assumptions for $DF_m|prmu;\ no_idle|C_{max}$ problem and then explains the position-based (PB) model, improved version of the position-based (IV_PB) model and sequence-based (SB) model, respectively.

- The problem involves F identical factories ($F > 1$) situated in different areas, collectively responsible for processing a set of N jobs.
- Factories operate under the same workflows, with each featuring a flowshop production system and a fixed sequence of M machines.
- Each job must be allocated to one of the F identical factories for processing and all jobs are processed sequentially on the M machines, maintaining the same order without interruption.
- Each machine must operate continuously without idle time between consecutive jobs.

- All jobs are ready from the beginning, and the initiation of the first job on each machine can be delayed to eliminate periods of inactivity.
- Jobs are processed one by one on each machine, with identical processing time across all factories.
- Interruptions or machine breakdowns are not considered in the model.
- The aim is to distribute jobs across factories and arrange their processing order in a way that minimizes the makespan (C_{max}).

Table 1 lists the parameters and indices that are frequently employed in the models.

2.1. Position-based (PB) model

PB model is the only model available in the literature for addressing $DF_m|prmu;\ no_idle|C_{max}$, which was introduced by Ying et al. [9]. PB model employs binary and continuous decision variables.

$X_{j,k,f} \rightarrow$ A binary decision variable that is 1 when job j is placed in position k in factory f, and 0 if not.

$C_{k,i,f} \rightarrow$ The completion time of the job at position k on machine i in factory f is represented by this continuous decision variable.

$$Minimize C_{max} \tag{1}$$

$$\sum_{k=1}^{N}\sum_{f=1}^{F} X_{j,k,f} = 1 \quad j \in \{1, \ldots, N\} \tag{2}$$

$$\sum_{j=1}^{N}\sum_{f=1}^{F} X_{j,k,f} = 1 \quad k \in \{1, \ldots, N\} \tag{3}$$

$$C_{1,\ i,f} = \sum_{j=1}^{N} X_{j,1,f} \times \ P_{j,i}\ f \in \{1, \ldots, F\};\ i \in \{1, \ldots, M\} \tag{4}$$

$$C_{k,\ i,f} = C_{k-1,\ i,f} + \sum_{j=1}^{N} X_{j,k,f} \times \ P_{j,i}\ f \in \{1, \ldots, F\};\ i \in \{1, \ldots, M\};\ k \in \{2, \ldots, N\} \tag{5}$$

$$C_{k,\ 1,f} \geq \sum_{j=1}^{N} X_{j,k,f} \times \ P_{j,1}\ f \in \{1, \ldots, F\};\ k \in \{1, \ldots, N\} \tag{6}$$

$$C_{k,\ i,f} \geq C_{k,\ i-1,f} + \sum_{j=1}^{N} X_{j,k,f} \times \ P_{j,i}\ f \in \{1, \ldots, F\};\ i \in \{2, \ldots, M\};\ k \in \{1, \ldots, N\} \tag{7}$$

$$C_{max} \geq C_{k,\ i,f}\ f \in \{1, \ldots, F\};\ i \in \{1, \ldots, M\};\ k \in \{1, \ldots, N\} \tag{8}$$

$$C_{k,i,f} \geq 0\ f \in \{1, \ldots, F\};\ i \in \{1, \ldots, M\};\ k \in \{1, \ldots, N\} \tag{9}$$

$$X_{j,k,f} \in \{0,\ 1\}\ f \in \{1, \ldots, F\};\ j,k \in \{1, \ldots, N\} \tag{10}$$

Table 1
Model parameters and indices.

Model parameter	Explanation
F	Total factories available
N	Total jobs available
M	Total machines available
j, k, p	Jobs indices (or positions in the job sequence); $j, k, p \in \{1, 2,..., N\}$
f	Factory indices; $f \in \{1, 2,..., F\}$
i	Machine indices; $i \in \{1, 2,...,M\}$
$P_{j,i}$	The time required to process job j on machine i
ω	A significantly large positive number

Constraint set (1) defines the makespan, representing the maximum time required to complete all jobs across the factories. Constraint set (2) addresses job assignments, ensuring that each job j is assigned to exactly one position k within a given factory f. Constraint set (3) concerns position occupancy, guaranteeing that each position k in all factories is occupied by exactly one job. Constraint set (4) defines the completion time of the first job on machine i in factory f, based on the processing times of the assigned jobs. Constraint set (5) handles the completion times of the following jobs, ensuring that the no-idle condition is satisfied by calculating the completion time of job k on machine i in factory f, considering the previous job's completion time. Constraint set (6) ensures that the completion time on the first machine in factory f is determined by the processing times of the assigned jobs. Constraint set (7) requires the completion time on machine i in factory f to include the completion time from the preceding machine. Constraint set (8) calculates the makespan, ensuring it is at least as large as the completion time for any job on any machine. Constraint set (9) enforces non-negativity for all completion times, preventing negative values. Finally, constraint set (10) defines $X_{j,k,f}$ as a binary variable.

2.2. Improved version of the position-based (IV_PB) model

The PB model allows for the opening of a number of positions that corresponds to the total number of jobs in each factory. However, it does not ensure that jobs are assigned to these positions in a sequential manner. For instance, take a case where $F = 2$ and $N = 6$; a possible solution according to the PB model might be $X_{3,1,1} = X_{1,3,1} = X_{4,1,2} = X_{2,2,2} = X_{5,4,2} = X_{6,5,2} = 1$. In this case, jobs 1 and 3 are assigned to factory 1 in the order {3, 1}, while jobs 2, 4, 5, and 6 are assigned to factory 2 in the order {4, 2, 5, 6}. Specifically, jobs 1 and 3 fill positions 1 and 3 in factory 1, leaving positions 2 unoccupied. Similarly, in factory 2, jobs 4, 2, 5, and 6 are allocated to positions 1, 2, 4, and 5, while positions 3 remain empty. This example highlights that PB model does not guarantee a sequential assignment, resulting in potential gaps between assigned positions. Constraint set (3) in the PB model makes certain that exactly N positions are filled from a total of $N \times F$ available positions. However, it does not guarantee that these filled positions are in a sequential order. Our aim is to design a model that guarantees jobs are assigned to positions in a sequential order. In light of the previous example, we want to guarantee that jobs 3 and 1 in the first factory are assigned to positions 1 and 2, and that jobs 4, 2, 5, and 6 in the second factory are assigned to positions 1, 2, 3 and 4 as well. To implement this, the model is modified by removing Constraint set (3) and introducing Constraint sets (11) and (12), which are described below.

$$\sum_{j=1}^{N} X_{j,k,f} \leq 1\ f \in \{1, \ldots, F\};\ k \in \{1, \ldots, N\} \tag{11}$$

$$\sum_{j=1}^{N} X_{j,k,f} \geq \sum_{j=1}^{N} X_{j,k+1,f}\ f \in \{1, \ldots, F\};\ k \in \{1, \ldots, (N-1)\} \tag{12}$$

Constraint set (11) specifies that each position k in every factory f can be assigned to no more than one job. It ensures that each position within a factory is occupied by only one job, preserving the integrity of the scheduling. Constraint set (12) establishes a relationship between consecutive positions in the schedule. Specifically, it ensures that if a job occupies position k, then at least as many jobs must occupy position k+1. This prevents scenarios where jobs could be skipped between positions, promoting a continuous flow of job processing.

Equation (1)

Constraint sets (2), (5), (7), (9), (10), (11), (12)

$$C_{1,\ 1,f} = \sum_{j=1}^{N} X_{j,1,f} \times \ P_{j,1}\ f \in \{1, \ldots, F\} \tag{13}$$

$$C_{max} \geq C_{N,\ M,f}\ f \in \{1,\ \ldots, F\} \tag{14}$$

Constraint set (4) in the PM model calculates the job's completion time for the first position on each machine m in all factories f. In IV_PB model, as specified in Constraint set (13), it is adequate to evaluate the completion time of the job in the first position exclusively on the first machine for each factory f. Constraint set (14) guarantees that the maximum completion time C_{max} is no smaller than the completion time of the final job on the last machine in each factory. While Constraint set (14) requires only F different constraints to determine the C_{max} value, Constraint set (8) requires $N \times M \times F$ different constraints. In addition, due to its more compact structure, IV_PB model does not require the Constraint set (6).

In the IV_PB model, decision variables $X_{j,k,f}$ (where $j,\ k \in \{1, \ldots, N\}$ and $f \in \{1, \ldots, F\}$) include a factory index. Since multiple identical factories exist, a feasible schedule can be assigned in $F!$ ways. To handle this symmetry, different inequality sets can be used [48], but not simultaneously. Below, the symmetry breaking inequalities called SBC1, SBC2, SBC3 and SBC4 are given and explained in detail, respectively.

Constraint set (15) (called SBC1) addresses symmetry by introducing a hierarchical order for assigning jobs to factories [49]. It states that if job p is allocated to factory f, then at least one job j (where $j \in \{1,\ldots,N\}$ and $j<p$) must be completed in factory $f-1$. This condition guarantees that the processing of subsequent jobs is linked to the execution of earlier jobs in the previous factory, ensuring a coherent flow of operations between factories.

$$\sum_{k=1}^{N} X_{p,k,f} \leq \sum_{j=1}^{p-1} \sum_{k=1}^{N} X_{j,k,f-1}\ f \in \{2,\ \ldots, F\};\ p \in \{2,\ \ldots, N\} \tag{15}$$

An alternative hierarchical set of constraints [48] can be utilized as specified in Constraint set (16) (called SBC2). This constraint set relates the jobs assigned to job p in factory f to the number of factories. On the left side, the total number of jobs assigned to job p is multiplied by the number of preceding factories ($f-1$). This indicates that the assignment of jobs in factory f must not exceed the total number of jobs assigned in previous factories (from 1 to $f-1$) for earlier jobs j (from 1 to $p-1$). In essence, the total number of jobs for job p in factory f cannot surpass the cumulative assignments from the earlier factories.

$$\sum_{k=1}^{N} X_{p,k,f} \times\ (f-1) \leq \sum_{j=1}^{p-1} \sum_{k=1}^{N} \sum_{s=1}^{f-1} X_{j,k,f}\ f \in \{2,\ \ldots, F\};\ p \in \{2,\ \ldots, N\} \tag{16}$$

To break the symmetry, the lexicographic ordering constraints [50–53] presented in Constraint set (17) (called SBC3) can be implemented. It compares the job assignments between two consecutive factories, f and $f+1$. On the left side, it calculates the total number of jobs assigned in factory f, with a weighting factor of $2^{(p-j)}$ that prioritizes later jobs j (jobs closer to p). The right side does the same for factory $f+1$. The key point is that the total weighted job assignments in factory f must be at least as large as those in factory $f+1$. This guarantees that there isn't a substantial decrease in work assignments when transitioning from one factory to another, promoting a more balanced allocation of tasks.

$$\sum_{j=1}^{p} \sum_{k=1}^{N} 2^{p-j} \times X_{j,k,f} \geq \sum_{j=1}^{p} \sum_{k=1}^{N} 2^{p-j} \times X_{j,k,f+1}\ f \in \{1,\ \ldots, F-1\};\ p \in \{1,\ \ldots, N\} \tag{17}$$

Rather than setting constraints for each job cardinality level, another lexicographic ordering constraints [48] described in Constraint set (18) (called SBC4) can be employed to define a specific order that includes all jobs.

$$\sum_{j=1}^{N} \sum_{k=1}^{N} 2^{N-j} \times X_{j,k,f} \geq \sum_{j=1}^{N} \sum_{k=1}^{N} 2^{N-j} \times X_{j,k,f+1}\ f \in \{1,\ \ldots, F-1\} \tag{18}$$

2.3. Sequence-based (SB) model

SB model is capable of addressing $DF_m|prmu;\ no_idle|C_{max}$ without needing to explicitly index factories. This model requires the following decision variables:

$X_{k,j} \rightarrow$ A binary decision variable that is 1 if job j is scheduled right after job k, and 0 otherwise.

$C_{j,i} \rightarrow$ The completion time of job j on machine i is represented by this continuous decision variable.

Equation (1)

$$\sum_{k=0|j\neq k}^{N} X_{k,j} = 1\ j \in \{1,\ \ldots, N\} \tag{19}$$

$$\sum_{j=0,\ k\neq j}^{N} X_{k,j}\ \leq 1\ k \in \{1,\ \ldots, N\} \tag{20}$$

$$\sum_{j=1}^{N} X_{0,j}\ = F \tag{21}$$

$$\sum_{k=1}^{N} X_{k,0} = F - 1 \tag{22}$$

$$X_{k,j} + \ X_{j,\ k}\ \leq 1\ j \in \{1,\ \ldots, N\} |\ k \neq j,\ j \rangle k;\ k \in \{1,\ \ldots, N-1\} \tag{23}$$

$$C_{j,1}\ \geq\ P_{j,1}\ j \in \{1,\ \ldots, N\} \tag{24}$$

$$C_{j,i}\ \geq\ C_{j,\ i-1} + P_{j,i}\ j \in \{1,\ \ldots, N\};\ i \in \{2,\ \ldots, M\} \tag{25}$$

$$C_{j,i} = \ C_{k,\ i} + P_{j,i} + \omega \times \left(X_{k,j} - 1\right)\ i \in \{1,\ \ldots, M\};\ j,k \in \{1,\ \ldots, N\}\ |\ j \neq k \tag{26}$$

$$C_{max}\ \geq C_{j,\ M}\ j \in \{1,\ \ldots, N\} \tag{27}$$

$$C_{j,i}\ \geq 0\ i \in \{1,\ \ldots, M\};\ j \in \{1,\ \ldots, N\} \tag{28}$$

$$X_{k,j}\ \in \{0,\ 1\}\ j,k \in \{0, 1,\ \ldots, N\}\ |\ j \neq k \tag{29}$$

Constraint set (19) ensures that each job j is immediately preceded by exactly one other job k (where k equals 0, representing the starting point), ensuring that every job has a distinct predecessor. Constraint set (20) ensures that each job k can be followed by no more than one job j, preventing multiple jobs from being processed directly after the same job. Constraint set (21) mandates that the number of jobs starting immediately after the dummy job (0) must be equal to the number of factories F. Constraint set (22) ensures that dummy job 0 is a successor to exactly $F-1$ jobs. Constraint set (23) eliminates circular dependencies between jobs k and j, ensuring that if job j follows job k, then job k cannot also follow job j. Constraint set (24) specifies that job j's completion time on the first machine must be at least as large as its processing time $P_{j,1}$. Meanwhile, Constraint set (25) specifies that the completion time of job j on machine i must be no less than the total of its completion time on the previous machine and the processing time on the current machine. Constraint set (26) states that if job j follows job k, its processing on any machine i cannot begin until job k has finished its processing on that machine. Constraint set (27) ensures that the maximum completion time C_{max} is at least equal to the completion time of any job j on the last machine M. Finally, the decision variables are defined by Constraint sets (28) and (29).

The SB model employs a dummy job 0 to split the full sequence into F segments, each corresponding to a distinct factory. This segmentation is carried out by embedding F repetitions of the dummy job 0 at various positions in the sequence, alongside the actual jobs. As a result, the model explores a search space of size $N+F$ positions. For instance, if $N = 6$ and $F = 2$, a possible solution might be represented by $X_{0,6} = X_{6,3} =$

$X_{3,0} = X_{0,1} = X_{1,4} = X_{4,2} = X_{2,5} = 1$, resulting in the sequence {0, 6, 3, 0, 1, 4, 2, 5}. In this instance, jobs 3 and 6 are assigned to factory 1 in the sequence {6, 3}, whereas the other jobs are assigned to factory 2 in the sequence {1, 4, 2, 5}.

3. The procedure for determining the upper bound value in the IV_PB model

As can be seen from the Constraint set (10), both PB and IV_PB models require a total of $N \times N \times F$ number of decision variables $X_{j,k,f}$ (where $f \in \{1, \ldots, F\}$ and $j, k \in \{1, \ldots, N\}$) to solve a given instance. As an example, with N = 10 and F = 3, the model will involve 300 activated decision variables $X_{j,k,f}$. Therefore, each model will contain 100 decision variables for every factory index f. However, in the given example, there are only 10 jobs available. If we assume $F = 1$ instead of $F = 3$, this results in 90 positions being unfilled after 10 jobs are allocated. In the case of $F = 3$, there will be 290 empty positions. A similar situation is valid for the decision variables $C_{k,i,f}$ (where $k \in \{1, \ldots, N\}$, $i \in \{1, \ldots, M\}$ and $f \in \{1, \ldots, F\}$) given in the Constraint set (9), where k index indicates position.

The proposed IV_PB model assigns jobs to positions sequentially. In other words, in any factory f, no job can be assigned to position k unless a job is previously assigned to position p, with the condition $p<k$. By taking advantage of this feature of IV_PB model, completely new mathematical model has been designed to determine the maximum limit for the number of positions within the decision variables of the IV_PB model, with further details provided below. To achieve this, a heuristic solution value (denoted as h_C_{max}) is first determined for a given instance using the proposed GRASP, which will be detailed in the following sections. The h_C_{max} value essentially represents the completion time of the factory with the longest completion time among the F factories. The proposed mathematical model seeks to identify the maximum number of jobs that can be allocated to a single factory, taking into account all jobs in a specific instance, and ensuring that the completion time of the last assigned job does not exceed h_C_{max} value determined for that instance. The decision variables of this model, which aims to maximize the number of jobs assigned to a single factory, based on the given heuristic solution value (h_C_{max}), are presented below.

$X_{j,k} \rightarrow$ Binary decision variable that is 1 when job j is placed in position k, and 0 if not.

$C_{k,i} \rightarrow$ The completion time of the job at position k on machine i is denoted by this continuous decision variable.

$$\text{Maksimize } Z = \sum_{j=1}^{N} \sum_{k=1}^{N} X_{j,k} \tag{30}$$

$$\sum_{k=1}^{N} X_{j,k} \leq 1 \; j \in \{1, \ldots, N\} \tag{31}$$

$$\sum_{j=1}^{N} X_{j,k} \leq 1 \; k \in \{1, \ldots, N\} \tag{32}$$

$$\sum_{j=1}^{N} X_{j,k} \geq \sum_{j=1}^{N} X_{j,k+1} \; k \in \{1, \ldots, N-1\} \tag{33}$$

$$C_{1,\,1} = \sum_{j=1}^{N} X_{j,1} \times P_{j,1} \tag{34}$$

$$C_{k,\,i} = C_{k-1,\,i} + \sum_{j=1}^{N} X_{j,k} \times P_{j,i} \; i \in \{1, \ldots, M\}; \; k \in \{2, \ldots, N\} \tag{35}$$

$$C_{k,\,i} \geq C_{k,\,i-1} + \sum_{j=1}^{N} X_{j,k} \times P_{j,i} \; i \in \{2, \ldots, M\}; \; k \in \{1, \ldots, N\} \tag{36}$$

$$C_{k,M} \leq h_Cmax \; k \in \{1, \ldots, N\} \tag{37}$$

$$C_{k,i} \geq 0 \; i \in \{1, \ldots, M\}; \; k \in \{1, \ldots, N\} \tag{38}$$

$$X_{j,k} \in \{0, 1\} \; k,j \in \{1, \ldots, N\} \tag{39}$$

The objective function (Constraint set (31)) seeks to maximize the total number of jobs assigned to a factory, represented by Z. Constraint set (31) ensures that each job is assigned to at most one position within the factory. Constraint sets (32) to (36) handle cases where the factory index is not included in Constraint sets (11), (12), (13), (5), and (7), respectively. Constraint set (37) limits the job's completion time at position k, ensuring it doesn't surpass the value of h_Cmax. Finally, Constraint sets (38) and (39) specify the acceptable ranges for the decision variables in the problem.

4. RIV_PB model generated by limiting the number of positions in IV_PB model

RIV_PB model is formulated by limiting the upper bounds on the number of positions for both binary, $X_{j,k,f}$ (where $f \in \{1, \ldots, F\}$ and $k, j \in \{1, \ldots, N\}$), and continuous, $C_{k,i,f}$ (where $f \in \{1, \ldots, F\}$; $i \in \{1, \ldots, M\}$; $k \in \{1, \ldots, N\}$), decision variables, to the Z value determined in the previous section. RIV_PB model described below, in contrast to IV_PB model, limits the total number of positions that can be utilized in the factories.

Equation (1)

$$\sum_{k=1}^{Z} \sum_{f=1}^{F} X_{j,k,f} = 1 \;\; j \in \{1, \ldots, N\} \tag{40}$$

$$\sum_{j=1}^{N} X_{j,k,f} \leq 1 \; f \in \{1, \ldots, F\}; \; k \in \{1, \ldots, Z\} \tag{41}$$

$$\sum_{j=1}^{N} X_{j,k,f} \geq \sum_{j=1}^{N} X_{j,k+1,f} \; f \in \{1, \ldots, F\}; \; k \in \{1, \ldots, (Z-1)\} \tag{42}$$

$$C_{1,\,1,f} = \sum_{j=1}^{N} X_{j,1,f} \times P_{j,1} \; f \in \{1, \ldots, F\} \tag{43}$$

$$C_{k,\,i,f} = C_{k-1,\,i,f} + \sum_{j=1}^{N} X_{j,k,f} \times P_{j,i} \; f \in \{1, \ldots, F\}; \; i \in \{1, \ldots, M\}; \; k \in \{2, \ldots, Z\} \tag{44}$$

$$C_{k,\,i,f} \geq C_{k,\,i-1,f} + \sum_{j=1}^{N} X_{j,k,f} \times P_{j,i} \; f \in \{1, \ldots, F\}; \; i \in \{2, \ldots, M\}; \; k \in \{1, \ldots, Z\} \tag{45}$$

$$C_{max} \geq C_{Z,\,M,f} \; f \in \{1, \ldots, F\} \tag{46}$$

$$C_{k,i,f} \geq 0 \; f \in \{1, \ldots, F\}; \; i \in \{1, \ldots, M\}; \; k \in \{1, \ldots, Z\} \tag{47}$$

$$X_{j,k,f} \in \{0, 1\} \; f \in \{1, \ldots, F\}; \; k \in \{1, \ldots, Z\}; \; j \in \{1, \ldots, N\} \tag{48}$$

5. The proposed Benders decomposition algorithms for solving DNIPFSP

The Benders decomposition algorithm, originally introduced by Benders [54], is a cutting-plane method widely used to solve complex combinatorial optimization problems [55]. It decomposes the original problem into two interconnected models: the master model (MM), derived from the master problem (MP), and the submodel (SM), derived from the subproblem (SP). The MM consists of a subset of the original

problem's variables and constraints, defining high-level scheduling and assignment decisions. The SP is constructed by fixing the variables determined by the MM solution and incorporating detailed feasibility and timing constraints. The solution process is iterative. The MM is first solved, providing values for its decision variables. These values are then used to construct and solve the SM. If the SM is infeasible, this implies that the MM's solution is also infeasible for the original problem. In such cases, a feasibility cut is generated from the dual of the SM and added to the MM to eliminate that infeasible solution. If the SM is feasible, an optimality cut is applied to refine the MM's solution space.

A key advantage of Benders decomposition is its ability to handle large-scale problems, as both the MM and the SM are typically easier to solve than the original problem. This often results in faster convergence, even though multiple iterations may be required [56]. A significant body of research has explored various strategies to enhance the efficiency and performance of the Benders decomposition algorithm, demonstrating that modifying specific components of the algorithm can yield substantial performance improvements (e.g., [48,57–59]).

This paper proposes an enhancement to the Benders decomposition algorithm by introducing an additional cut in each iteration to improve efficiency. To achieve this, we integrate the traditional Benders decomposition algorithm with GRASP. In this section, we first apply the basic form of Benders decomposition to the RIV_PB model discussed earlier. Then, we introduce a hybrid version of the algorithm aimed at accelerating the convergence of the original method.

5.1. The proposed pure Benders decomposition algorithm

In this section, we introduce a pure Benders decomposition algorithm developed on the RIV_PB model, incorporating constraints on the order of positions and an upper limit on the total number of available positions. Let M(C,X) represent the formulation defined by equations (40)-(48), where $X = \{X_{j,k,f} | j = 1, \ldots, N; k = 1, \ldots, Z; f = 1, \ldots, F\}$ and $C = \{C_{k,i,f} | k = 1, \ldots, Z; i = 1, \ldots, M; f = 1, \ldots, F\}$ denote the decision variable vectors. Suppose that the variables X have already been fixed as $X = \widehat{X} = \{X \mid X \text{ satisfies } (40), (41), (42) \text{ and } (48)\}$. This yields the following Benders Subproblem (SP).

Minimize C_{max}

$$C_{1,\,1,f} = \sum_{j=1}^{N} \widehat{X}_{j,1,f} \times P_{j,1} \; f \in \{1, \ldots, F\} \tag{43a}$$

$$C_{k,\,i,f} = C_{k-1,\,i,f} + \sum_{j=1}^{N} \widehat{X}_{j,k,f} \times P_{j,i} \; f \in \{1, \ldots, F\}; \; i \in \{1, \ldots, M\}; \; k \in \{2, \ldots, Z\} \tag{44a}$$

$$C_{k,\,i,f} \geq C_{k,\,i-1,f} + \sum_{j=1}^{N} \widehat{X}_{j,k,f} \times P_{j,i} \; f \in \{1, \ldots, F\}; \; i \in \{2, \ldots, M\}; \; k \in \{1, \ldots, Z\} \tag{45a}$$

$$C_{max} \geq C_{Z,\,M,f} \; f \in \{1, \ldots, F\} \tag{46a}$$

$$C_{k,i,f} \geq 0 \; f \in \{1, \ldots, F\}; \; i \in \{1, \ldots, M\}; \; k \in \{1, \ldots, Z\} \tag{47a}$$

The resulting SP formulation, $M(C, \widehat{X})$, includes only the variables C and consists of constraint sets associated with the dual variables $\alpha = \{\alpha_f = unrestricted \mid f = 1, \ldots, F\}$ corresponding to Constraint set (43), $\gamma = \{\gamma_{k,i,f} = unrestricted \mid k = 2, \ldots, Z; i = 1, \ldots, M; f = 1, \ldots, F\}$ corresponding to Constraint set (44), $\beta = \{\beta_{k,i,f} \geq 0 \mid k = 1, \ldots, Z; i = 2, \ldots, M; f = 1, \ldots, F\}$ corresponding to Constraint set (45), and $\delta = \{\delta_f \geq 0 \mid f = 1, \ldots, F\}$ corresponding to Constraint set (46). The dual of $M(C, \widehat{X})$, denoted by $D = (\alpha, \gamma, \beta, \delta, \widehat{X})$, corresponds to the Submodel (SM) and is formulated as follows:

$$\text{Maximize} \sum_{k=1}^{Z} \sum_{i=2}^{M} \sum_{f=1}^{F} \beta_{k,i,f} \sum_{j=1}^{N} \widehat{X}_{j,k,f} P_{j,i} + \sum_{k=2}^{Z} \sum_{i=1}^{M} \sum_{f=1}^{F} \gamma_{k,i,f} \sum_{j=1}^{N} \widehat{X}_{j,k,f} P_{j,i} + \sum_{f=1}^{F} \alpha_f \sum_{j=1}^{N} \widehat{X}_{j,1,f} P_{j,1} \tag{49}$$

$$\alpha_f - \beta_{1,2,f} - \gamma_{2,1,f} \leq 0 \; f \in \{1, \ldots, F\} \tag{50}$$

$$\beta_{1,i,f} - \beta_{1,i+1,f} - \gamma_{2,i,f} \leq 0 \; i \in \{2, \ldots, (M-1)\}; \; f \in \{1, \ldots, F\} \tag{51}$$

$$\beta_{1,M,f} - \gamma_{2,M,f} \leq 0 \; f \in \{1, \ldots, F\} \tag{52}$$

$$\gamma_{Z,1,f} - \beta_{Z,2,f} \leq 0 \; f \in \{1, \ldots, F\} \tag{53}$$

$$\beta_{Z,i,f} - \beta_{Z,i+1,f} + \gamma_{Z,i,f} \leq 0 \; i \in \{2, \ldots, (M-1)\}; f \in \{1, \ldots, F\} \tag{54}$$

$$\beta_{Z,M,f} + \gamma_{Z,M,f} - \delta_f \leq 0 \; f \in \{1, \ldots, F\} \tag{55}$$

$$\gamma_{k,1,f} - \gamma_{k+1,1,f} - \beta_{k,2,f} \leq 0 \; k \in \{2, \ldots, (Z-1)\}; f \in \{1, \ldots, F\} \tag{56}$$

$$\beta_{k,M,f} + \gamma_{k,M,f} - \gamma_{k+1,M,f} \leq 0 \; k \in \{2, \ldots, (Z-1)\}; f \in \{1, \ldots, F\} \tag{57}$$

$$\beta_{k,i,f} - \beta_{k,i+1,f} + \gamma_{k,i,f} - \gamma_{k+1,i,f} \leq 0 \; k \in \{2, \ldots, (Z-1)\}; i \in \{2, \ldots, (M-1)\}; f \in \{1, \ldots, F\} \tag{58}$$

$$\sum_{f=1}^{F} \delta_f \leq 1 \tag{59}$$

By incorporating Constraint sets (40),(41),(42), and (48), the MM consistently generates a feasible solution. In other words, our formulation inherently ensures that the MM is always feasible. Unlike traditional Benders decomposition approaches, where infeasible solutions may arise and require feasibility cuts, our approach eliminates the need for such cuts, as every solution produced by the MM can always be processed by the SM without infeasibility concerns. Since the feasibility of the MM is always guaranteed, the role of the SM is not to eliminate infeasible solutions but rather to iteratively refine the optimality conditions. This implies that $(\alpha, \gamma, \beta, \delta, \widehat{X})$ is feasible for a given $\widehat{X}$. Once the optimal solution $(\widehat{\alpha}, \widehat{\gamma}, \widehat{\beta}, \widehat{\delta})$ for the dual problem, namely SM, is obtained, the coefficient matrix for the variables is computed by following the steps outlined in Algorithm 1.

Subsequently, the Benders optimality cuts, as specified in Constraint set (60), are generated.

$$\mu \geq \sum_{j=1}^{N} \sum_{k=1}^{Z} \sum_{f=1}^{F} \text{CoeffCut}\,(j,k,f) \times X_{j,k,f} \tag{60}$$

In Constraint set (60), μ is used to represent the lower bound of the optimal value for $M(C, X)$. Having obtained this result, the following reformulation of M(C,X) is presented, designated as the master model (MM), as presented below:

$$\text{Minimize } \mu \tag{61}$$

Constaint sets (40),(41),(42),(48) and (60)

The MM, which includes multiple optimality cuts, is well-suited for solution via a cutting-plane algorithm. The MM is solved without Constraint 60 in the first iteration. As the algorithm progresses, necessary cuts are dynamically generated and incorporated into the model. The iterative procedure concludes when a specific MM(Cut) is solved, where Cut represents a subset of the overall set Cut_D.

Algorithm 1
The precedure for calculating the dual variables' coefficient matrix.

1: *CoeffCut*(*j, k, f*) ← 0 ∀*j* ∈ *N*, ∀*k* ∈ *Z*, ∀*f* ∈ *F* // the dual variables' coefficient matrix
2: for (*f* ← 1 to F)
3: for (*j* ← 1 to *N*)
4: if ($\widehat{\alpha}_f$ is greater than 0)
5: *CoeffCut* (*j*,1,*f*) ← ($\widehat{\alpha}_f \times P_{j,1}$) + *CoeffCut* (*j*,1,*f*)
6: endif
7: for (*k* ← 1 to *Z*)
8: for (*i* ← 1 to *M*)
9: if ($\widehat{\beta}_{k,i,f}$ is greater than 0 && *i* is greater than or equal to 2)
10: *CoeffCut* (*j*,*k*,*f*) ← ($\widehat{\beta}_{k,i,f} \times P_{j,i}$) + *CoeffCut* (*j*,*k*,*f*)
11: endif
12: if ($\widehat{\gamma}_{k,i,f}$ is greater than 0 && *k* is greater than or equal to 2)
13: *CoeffCut* (*j*,*k*,*f*) ← ($\widehat{\gamma}_{k,i,f} \times P_{j,i}$) + *CoeffCut* (*j*,*k*,*f*)
14: endif
15: endfor
16: endfor
17: endfor
18: if ($\widehat{\delta}_f$ is greater than 0)
19:for (j ← 1 to *N*)
20:for (k ← 1 to *Z*)
21:*CoeffCut* (*j*,*k*,*f*) ← $\widehat{\delta}_f$ + *CoeffCut* (*j*,*k*,*f*)
22:endfor
23:endfor
24:endif
25: endfor

5.2. The proposed hybrid Benders decomposition algorithm

This section presents a hybrid Benders decomposition algorithm designed to accelerate the convergence of the pure Benders algorithm towards an optimal solution. The proposed hybrid algorithm is a combination of GRASP and the pure Benders algorithm given in the preceding subsection. The operation of the hybrid algorithm is based on deriving two separate optimality cuts from the optimal solution provided by the master model (MM) at each iteration and reintegrating these cuts into MM to generate the next solution. One optimality cut is generated by applying the optimal solution $\widehat{X}$ from MM directly to the submodel (SM) and solving it. The other cut is derived by using this optimal solution $\widehat{X}$ as the initial solution for GRASP. This solution is then improved by GRASP, and the improved solution $\widehat{X}_{heuristic}$, is used to solve SM. In essence, the hybrid Benders decomposition introduces an extra optimality cut into MM during each iteration.

Metaheuristic algorithms require computation time, and the quality of the solution provided to SM typically depends on the runtime of the metaheuristic. Therefore, it's crucial to strike a balance between solution quality and computation time. In the proposed hybrid Benders algorithm, to maintain this balance, the initial solution generation phase of GRASP is skipped when a new cut is generated (as the initial solution is already derived from MM). Afterward, GRASP runs for just one iteration. Algorithm 2 provides the pseudo-code that outlines the hybrid Benders decomposition algorithm 2.

Algorithm 2
The main logic of the hybrid Benders decomposition algorithm.

Input: Problem data, allowable optimality gap $\varepsilon \geq 0$
1. Set *LowerBound* ← -∞, *UpperBound* ← ∞
2: while (*LowerBound* ≤ *UpperBound*) **do**
3:Execute the MM to obtain $\widehat{X} = \{X | X \text{ satisfies } (40), (41), (42), \text{ and } (48)\}$
4: *obj_mm* ← Objective function value of MM solution (μ)
5.if (*LowerBound* < *obj_mm*)
6:*LowerBound* ← *obj_mm*
7:endif
8:Execute SM with Constraint sets (49) – (59) and $\widehat{X}$, and generate an optimality cut (*Cut* 1)
using this SM solution and Constraint set (60)
9: *obj_sub* ← Objective function value of SM solution
10:if (*UpperBound* > *obj_sub*)
11:*UpperBound* ← *obj_sub;*
12:endif
13: By providing MM solution ($\widehat{X}$) as the initial solution for GRASP, execute GRASP for just one iteration to obtain a new solution ($\widehat{X}_{heuristic}$)
14: Execute SM with Constraint sets (49) – (59) with $\widehat{X}_{heuristic}$, and generate an additional
optimality cut (*Cut* 2) using this SM solution and Constraint set (60)
15: *obj_sub_heuristic* ← Objective function value of SM solution
16:if (*UpperBound* > *obj_sub_heuristic*)
17:*UpperBound* ← *obj_sub_heuristic*
18:endif
19: Insert the generated optimality cuts, *Cut* 1 and *Cut* 2, into MM
20: endwhile
21: Report the best solution and its value from the final MM run

6. Proposed GRASP for solving DNIPFSP

This section introduces the proposed Greedy Randomized Adaptive Search Procedure (GRASP) algorithm for solving the DNIPFSP. GRASP combines several advantages: (a) a greedy approach to generate good initial solutions, (b) randomization to explore different areas of the solution space, and (c) local search to refine solutions towards local optima [46,47,60]. It has proven to be a powerful metaheuristic for a variety of optimization problems [46,61–64]. A standard GRASP algorithm works iteratively and includes two main phases: construction and local search [47]. In the construction phase, a greedy method is applied to generate a feasible solution, while the local search phase improves the best solution found so far. To our knowledge, GRASP has never been applied to distributed flowshop scheduling problems.

The proposed GRASP algorithm begins by constructing an initial solution in the same way as standard GRASP. After this solution is generated, the modified LS3 local search algorithm [21] is applied to refine it further. Then, the core steps of the algorithm commence. A perturbation mechanism is used to enhance the solution obtained from the local search phase. This perturbation mechanism consists of two main steps: deconstruction and reconstruction, which involve removing and reinserting partial jobs into the solution. After each perturbation step, the LS3 algorithm is applied again to improve the new solution. These steps continue until the stopping criteria are met.

The main steps of the proposed GRASP algorithm are detailed in Algorithm 3. The algorithm takes three input parameters: α, which controls the balance between greediness and randomness, *d*, which specifies the total number of jobs to be removed during the destruction phase, and Grasp_Max_Step, which defines the maximum number of GRASP iterations. The specific details of the algorithm are explained in the following sections Algorithm 3.

Algorithm 3
The proposed GRASP.

Input: Problem data, a d, Grasp_Max step
Output: global best solution, π_{best}
1:$\pi_{initial}$ ← *Construction* (*JL*, a) //*JL(.)* represents the list of all available jobs
2:π_{best} ← $\pi_{initial}$, *obj*(π_{best}) ← *obj*($\pi_{initial}$)
3.*Grasp_Step* ← 0
4:While *Grasp_Step* ≤ *Grasp*_Max step (or a given computational time limit) **do**
5:$\pi_{initial}$ ← *Modified LS3 local search* ($\pi_{initial}$)
6:If *obj*($\pi_{initial}$) < *obj*(π_{best})
7:π_{best} ← $\pi_{initial}$
8: *obj*(π_{best}) ← *obj*($\pi_{initial}$)
9:Endif
10:$\pi_{incumbent}$ ← $\pi_{initial}$
11:$\pi_{partial}$, *EJL* ← *Destruction* ($\pi_{incumbent}$, *d*) //*EJL* represents the list of the extracted jobs
12:$\pi_{initial}$ ← *Reconstruction* ($\pi_{partial}$, *EJL*,a)
13.*Grasp_Step* ← *Grasp_Step* + 1
14:Endwhile
15. Report π_{best}

Problem data for simple example

Machines \ Jobs	J1	J2	J3	J4
M1	1	86	28	32
M2	4	21	67	17

Iteration 1:
Step 1: Initialize Gantt chart structure for each factory
Step 2: JL = {1, 2, 3, 4}
Step 3: CT_{111} = 5, CT_{121} = 5, CT_{211} = 107, CT_{221} = 107, CT_{311} = 95, CT_{321} = 95, CT_{411} = 49, CT_{421} = 49 (Since no jobs have been scheduled to the factories yet, there is only 1 available position in each factory)
Step 4. FL = {5, 5, 107, 107, 95, 95, 49, 49}
Step 5: ṣ = 5, ṡ =107, α = 0.5, g = 56, RCL = {CT_{111} = 5, CT_{121} = 5, CT_{411} = 49, CT_{421} = 49}
Step 6: Randomly select a member from RCL list (CT_{111} is selected; in other words, job 1 is scheduled to position 1 in factory 1)
Step 7: JL = {2, 3, 4}
Iteration 2:
Step 3: CT_{211} = 111, CT_{212} = 108, CT_{221} = 107, CT_{311} = 99, CT_{312} = 96, CT_{321} = 95, CT_{411} = 53, CT_{412} = 50 , CT_{421} = 49 (Since job 1 has already been scheduled to factory 1, there are two available positions for the next job in factory 1)
Step 4. FL = {111, 108, 107, 99, 96, 95, 53, 50, 49}
Step 5: ṣ = 49, ṡ =111, α = 0.5, g = 80, RCL = {CT_{411} = 53, CT_{412} = 50 , CT_{421} = 49}
Step 6: Randomly select a member from RCL list (CT_{421} is selected; in other words, job 4 is scheduled to position 1 in factory 2)
Step 7: JL = {2, 3}
Iteration 3:
Step 3: CT_{211} =111, CT_{212} = 108, CT_{221} = 135, CT_{222} = 139, CT_{311} = 99, CT_{312} = 96 , CT_{321} = 112, CT_{322} = 127 (Since jobs 1 and 4 have already been assigned to factories 1 and 2, respectively, both factories 1 and 2 have two available positions for the next job)
Step 4. FL = {111, 108, 135, 139, 99, 96, 112, 127}
Step 5: ṣ = 96, ṡ =139, α = 0.5, g = 117.50, RCL = {CT_{211} =111, CT_{212} =108, CT_{311} = 99, CT_{312} = 96, CT_{321} = 112}
Step 6: Randomly select a member from RCL list (CT_{212} is selected; in other words, job 2 is scheduled to position 2 in factory 1)
Step 7: JL = {3}
Iteration 4:
Step 3: CT_{311} =185, CT_{312} = 136, CT_{313} = 175, CT_{321} = 112 , CT_{322} = 127 (Since jobs 1 and 2 have already been scheduled for factory 1, there are three available positions for the next job in factory 1. Additionally, since job 4 has been scheduled for factory 2, there are two available positions for the new job in factory 2)
Step 4. FL = {185, 136, 175, 112, 127}
Step 5: ṣ = 112, ṡ =185, α = 0.5, g = 148.50, RCL = {CT_{312} = 136, CT_{321} = 112 , CT_{322} = 127}
Step 6: Randomly select a member from RCL list (CT_{321} is selected; in other words, job 3 is scheduled to position 1 in factory 2)
In factory 2, job 4, which is previously scheduled in position 1, has been moved to position 2.
Step 7: JL = {} (All jobs in JL are scheduled)
Iteration 5: π_1={1, 2}, π_2={3, 4}, $C^1_{\max}$ = 108, $C^2_{\max}$ = 112 and $C_{\max}$ = 112

Fig 1. Illustration of the initial solution construction procedure.

6.1. Construction phase of the proposed GRASP

The proposed GRASP algorithm for solving the DNIPFSP begins by inserting all jobs that need to be scheduled into a list called the jobs list (JL). Then, each job in JL is evaluated sequentially from left to right. For each job, trial schedules are generated considering all available factories. The evaluation of each job involves considering all available positions in each factory using an acceleration method. For each job j, when it is assigned to position p in factory f, the completion time, CT_{jfp}, is calculated. These completion times are then stored in a list called FL. This process continues until all jobs in JL are evaluated in this manner. The job selected for the next scheduling step is chosen by evaluating all jobs in FL with a greedy function. This function assesses the suitability of each job in FL for scheduling. Using the parameter (α), a restricted candidate list (RCL) is created, balancing greediness and randomness within the algorithm. The threshold value, denoted as g, is calculated as $g = ṣ + \alpha(ṡ - ṣ)$, where $ṣ$ represents the minimum value in FL and $ṡ$ is the maximum value. This threshold aids in constructing the RCL, which contains job-factory-position combinations in FL whose completion times do not exceed g. During each iteration, a job is randomly selected from the RCL, incorporating randomness into GRASP algorithm. Once a job is chosen, it is removed from JL, and FL is updated accordingly. The construction phase continues iteratively until all jobs in JL are assigned to factories. The construction phase consists of the following steps:

1. Initialize the Gantt chart structure for each factory.
2. Create the jobs list (*JL*) using all unscheduled jobs.
3. Temporarily schedule jobs in *JL* from left to right, considering all available factories and positions using the acceleration method. For each job in *JL*, determine the position p in factory f that minimizes the completion time, CT_{jfp}.
4. Insert the calculated CT_{jfp} for each job into a list called *FL*.
5. Determine the *RCL* by calculating the minimum $ṣ = \min\{CT_{jfp}$ (If all values are the same, select one at random)$\}$, the maximum $ṡ = \max\{CT_{jfp}$ (If all values are the same, select one at random)$\}$, and the threshold value $g = ṣ + \alpha(ṡ\text{-}ṣ)$. *RCL* is created by selecting job-factory-position combinations from *FL* where the completion time is less than or equal to g.
6. Select a member from the *RCL* randomly and add it to the schedule with the belonging attributes which are obtained in Step 3 (The factory and position where the selected job will be processed, along with its completion time details).
7. Delete the chosen job from *JL*. If all jobs in *JL* are scheduled, proceed to Step 8; otherwise, return to Step 3.
8. Calculate the objective function from the resulting factory schedules.

The initial solution construction process is illustrated in Fig 1, where an example with four jobs ($N = 4$), two factories ($F = 2$) each having two machines ($M = 2$) is shown. In this example, the α parameter is set to 0.5. This approach efficiently constructs an initial solution by combining greedy strategies, randomization, and local search, making it a powerful tool for solving DNIPFSP.

6.2. Modified LS3 local search algorithm

Ruiz et al. [21] introduced the LS3 local search algorithm for solving the DPFSP, which starts by exploring a feasible initial solution. It simultaneously considers both the factory assignment and job sequencing. The factory with the highest C_{max}, referred to as the critical factory, is identified. A job chosen at random from the factory is assessed in every potential position across all factories, even in the one that currently defines the makespan. In the event that any of these new positions result in a better C_{max}, the job is moved, and the search process begins again. If no improvement occurs, the job is reverted to its initial position, and the search continues. The process is executed without repetition and randomly, until every job from the critical factory has been tested. However, initial trials showed that focusing only on the critical factory's schedule may not be effective for intensification. It was also found that intensification could be greatly enhanced by broadening the search to include the schedules of other factories.

In this paper, the LS3 local search algorithm is enhanced (referred to as the modified LS3 local search) to not only focus on the critical factory but also improve the schedules of other factories. To achieve this, a factory selection rule based on the roulette wheel selection method is introduced. This method ensures that the modified LS3 algorithm gives preference to jobs from the critical factory, which has a higher probability of being selected compared to other factories. However, other factories (aside from the critical factory) are also given an opportunity to be selected, with their selection probability proportional to their C_{max} values. The pseudocode for the modified LS3 local search algorithm is provided in Algorithm 4.

As indicated in line 5 of Algorithm 4, a factory is selected, denoted as f_{sel} using the roulette wheel selection method [65]. This technique scales the maximum completion times for each factory, $\{C_{\max}(\pi_1), \{C_{\max}(\pi_2), \ldots, \{C_{\max}(\pi_F)\}$, ensuring that the total of the rescaled maximum completion times equals one. To perform the selection, a random number within the range [0, 1] is first generated (representing the spin of the wheel). Then, the first factory whose cumulative rescaled maximum completion time exceeds the generated number is chosen. The steps involved in the roulette wheel selection process are outlined as follows:

1. Compute the sum of the maximum completion times for all factories, referred to as F_{sum}.
2. Normalize the maximum completion times of each factory by dividing them by F_{sum} to calculate the expected value for each factory.
3. Generate a random number, R_s, within the interval [0, 1].
4. Iterate through the factories, accumulating their expected values, until the total reaches or exceeds R_s. The factory whose expected value causes the cumulative sum to surpass this threshold is the selected one.

After applying the roulette wheel selection method, a job, j, is randomly removed from the selected factory, f_{sel}, (line 6), and $C_{\max}$ value of f_{sel} after job j is removed is calculated (line 7). Then, job j is tested in all available factories in order. When testing job j in the next factory f, $C_{\max}$ calculation is performed for all positions of the factory f using the acceleration method given in Section 6.4 (lines 9 and 10). As a result, the position p^f that gives the lowest $C_{\max}$ value, $C^f_{\max}$, for the factory f is obtained (line 11). The sum of the $C_{\max}$ values before job j is removed from factory f_{sel} and added to the best position p^f in factory f is calculated. A cost value, $Cost^f$, is then computed by subtracting this total makespan value from the total $C_{\max}$ values after the changes are made in factories f_{sel} and f (line 12). After that, the factory that gives the maximum cost value is determined, $f_{\text{cost_max}}$ (line 14). If $Cost^{f_{\text{cost_max}}}$ value is greater than 0, then job j is permanently placed in $p^{f_{\text{cost_max}}}$ position (line 16) of the factory $f_{\text{cost_max}}$. Then, in order to further improve the resulting schedule, one of the jobs in positions $p^{f_{\text{cost_max}}}$ - 1 or $p^{f_{\text{cost_max}}} + 1$ is removed and tested to all positions (lines 17-18). In the same way, the removed job is inserted into the best position of $f_{\text{cost_max}}$, which yields the lowest $C_{\max}$ value (line 19). The algorithm's counter is then set to 0 (line 21), and after the necessary updates are made (lines 26 and 27), the algorithm proceeds by selecting a new factory and randomly removing a job from the selected factory. If $Cost^{f_{\text{cost_max}}}$ is less than or equal to 0, the job is reinserted back into its original position and the algorithm's counter is increased by one (lines 23-24). The algorithm will continue running until its counter equals the cardinality of the factory (number of jobs in that factory) with the maximum $C_{\max}$ value at the start of the algorithm. In preliminary tests, it is observed that the modified LS3 algorithm provided a more balanced workload distribution between the factories Algorithm 4.

6.3. Destruction and reconstruction

In the destruction phase, the current solution, denoted as $\pi_{incumbent}$, undergoes partial removal of some of its components, leading to two partial permutations: one is a job list (*EJL*) consisting of the jobs that have been removed from $\pi_{incumbent}$ and another with the remaining jobs from the original solution ($\pi_{partial}$). The destruction operator utilized in this study, proposed by Ruiz et al. [21], has been shown to be effective in their research. This operator removes $d/2$ jobs randomly from the factory with the highest C_{max} value (if there are multiple factories with the highest C_{max}, no effort is made to break the tie). The remaining $d/2$ jobs are randomly selected from the other $F - 1$ factories, with the exception of the one having the highest C_{max}.

During the reconstruction phase, a heuristic, usually a greedy algorithm, is used to reinsert the removed jobs into the solution, thereby generating a new complete solution.The construction phase of the proposed GRASP serves as the reconstruction operator, where jobs from *EJL* list are progressively added to the partial schedule.

6.4. Makespan calculation and acceleration method

The determination of makespan values for neighborhoods is vital for enhancing the algorithm's efficiency. To comply with the no-idle condition, it is necessary to eliminate any idle periods between the

Algorithm 4
The modified LS3 algorithm.

Input: {π1, π2, ..., πF} //It starts with a feasible solution
1: $C^*_{\max} \leftarrow \max_{f=1}^{F} \{C_{\max}(\pi_1), \{C_{\max}(\pi_2), \ldots, \{C_{\max}(\pi_F)\}$
2: $f^* \leftarrow$ Factory with $C^*_{\max}$ value
3: Cnt ← 0
4: While (Cnt ≤ $|\pi_{f^*}|$) do
5:$f_{\text{sel}} \leftarrow$ Select a factory using the roulette wheel selection method
6:Randomly select, without repetition, a job j from position k of $\pi_{f_{sel}}$
7:$C^{f_{\text{sel}}}_{max} \leftarrow$ Calculate the makespan of f_{sel} after extracting job j
8:For (f ← 1to F)
9:Temporarily place job j in all possible positions of π_f and evaluate using the acceleration method
10:$C^f_{\max}$ is the lowest $C_{\max}$ obtained for factory f
11:pf denotes the position that results in the minimum $C_{\max}$ for factory f
12:$Cost^f \leftarrow (C_{\max}(\pi_{f_{\text{sel}}}) + C_{\max}(\pi_f))$-$(C^f_{\max} + C^{f_{\text{sel}}}_{max})$
13:Endfor
14:Fcoast max ←arg $\left(\max_{f=1}^{F} (Cost^f)\right)$
15:If ($Cost^{f_{\text{cost_max}}}$
16:Insert the job j into $\pi_{f_{\text{cost_max}}}$ at position $p^{f_{\text{cost_max}}}$ resulting in the minimum $C_{\max}$
17:Extract at random job h from position $p^{f_{\text{cost_max}}}$-1 or $p^{f_{\text{cost_max}}}$+1 from $\pi_{f_{\text{cost_max}}}$
18:Evaluate job h in all possible positions of $\pi_{f_{\text{cost_max}}}$ using acceleration method
19:$C_{\max}\left(\pi_{f_{\text{cost_max}}}\right) \leftarrow$ Insert the job h into $\pi_{f_{\text{cost_max}}}$ at position $p^{f_{\text{cost_max}}}$ resulting in the lowest $C_{\max}$
20:$C_{\max}(\pi_{f_{\text{sel}}}) \leftarrow C^{f_{\text{sel}}}_{max}$
21:Cnt = 0
22:Else
23:Return job j to position k of f_{sel}
24:Cnt ← Cnt + 1
25:Endif
26:$C^*_{\max} \leftarrow \max_{f=1}^{F} \{C_{\max}(\pi_1), \{C_{\max}(\pi_2), \ldots, \{C_{\max}(\pi_F)\}$
27:$f^* \leftarrow$ Factory with $C^*_{\max}$ value
28: Endwhile
29. Report {π1, π2, ..., πF} and $\{C_{\max}(\pi_1), \{C_{\max}(\pi_2), \ldots, \{C_{\max}(\pi_F)\}$

processing of consecutive jobs on each machine [26]. On the other hand, it is well known that flowshop problems exhibit a reversibility property [26,66]. This property allows the makespan of a job sequence π to be calculated by moving through the sequence from the first job to the last (called forward calculation) or in reverse, from the last job back to the first (called backward calculation). Suppose that when calculating in the forward (backward), $S_{[k],i}$ ($Sb_{[k],i}$) and $C_{[k],i}$ ($Cb_{[k],i}$) represent the starting and completing times of the job at position k on machine i, respectively. Calculation of C_{max} in the forward direction given in Algorithm 5.

Algorithm 5
Calculation of Cmax in forward direction.

1: $S_{[1],1} \leftarrow 0$ // The job occupying the first position on the first machine
2: $C_{[1],1} \leftarrow S_{[1],1} + P_{[1],1}$ // The job occupying the first position on the first machine
3: for ($i \leftarrow 2$; $i \leq M$) // The job occupying the first position on the remaining machines
4:$S_{[1],i} \leftarrow C_{[1],i-1}$
5:$C_{[1],i} \leftarrow S_{[1],i} + P_{[1],i}$
6: endfor
7: for ($k \leftarrow 2$; $k \leq N$) //Jobs placed in positions excluding position 1
8:$S_{[k],1} \leftarrow C_{[k-1],1}$ //first machine
9:$C_{[k],1} \leftarrow S_{[k],1} + P_{[k],1}$ //first machine
10:$S_{[k],2} \leftarrow$ **max** ($C_{[k-1],2}$, $C_{[k],1}$) //second machine
11:$C_{[k],2} \leftarrow S_{[k],2} + P_{[k],2}$ //second machine
12:a ← 0
13:a ← **max** ($C_{[k],1}$ - $C_{[k-1],2}$, 0)
14:for ($i \leftarrow 3$; $i \leq M$) //remaining machines
15:$S_{[k],i} \leftarrow$ **max** ($C_{[k-1],i}$ + a, $C_{[k],i-1}$)
16:$C_{[k],i} \leftarrow S_{[k],i} + P_{[k],i}$
17:a ← a + **max** ($C_{[k],i-1} - (C_{[k-1],i}$ + a), 0);
18:endfor
19: endfor
20: Return $C_{max} \leftarrow C_{[N],M}$

Calculation of C_{max} in the backward direction given in Algorithm 6.

Let us consider an example with three jobs and three machines, where jobs are denoted as $j \in \{1, \ldots, N\}$ and machines as $i \in \{1, \ldots, M\}$. The processing times of these three jobs on the three machines are as follows:

$$[\boldsymbol{P_{j,i}}]_{3\times3} = \begin{bmatrix} 5 & 8 & 6 \\ 7 & 4 & 5 \\ 6 & 9 & 7 \end{bmatrix}$$

Let us consider a schedule π = {1, 2, 3}. If we perform forward computation for this example problem, the starting and completion times of job $\pi_1 = \{1\}$ on the first machine are $S_{[1],1} = 0$ and $C_{[1],1} = 5$, respectively. On the second machine, these times are $S_{[1],2} = 5$ and $C_{[1],2} = 13$. Similarly, on the third machine, the starting and completion times are $S_{[1],3} = 13$ and $C_{[1],3} = 19$, respectively.

The next job in the sequence is $\pi_2 = \{2\}$. The starting and completion times on the first machine are computed as $S_{[2],1} = 5$ and $C_{[2],1} = 12$, respectively. On the second machine, the starting time is determined as $S_{[2],2} = \max(C_{[1],2}$ (13), $C_{[2],1}$ (12)) = 13 and the completion time is $C_{[2],2} = 17$. Before scheduling job 2 on the second machine, the total idle time is calculated as a = max($C_{[2],1}$ (12) - $C_{[1],2}$ (13), 0) = 0. On the third machine, the starting and completion times are $S_{[2],3} = \max(C_{[1],3}$ (19) + a (0), $C_{[2],2}$ (17)) = 19 and $C_{[2],3} = 24$, respectively. Before scheduling job 2 on the third machine, the total idle time is calculated as a = a (0) + max($C_{[2],2}$ (17) – ($C_{[1],3}$ (19)+ a (0)), 0) = 0.

The last job in the sequence is $\pi_3 = \{3\}$. The starting and completion times on the first machine are computed as $S_{[3],1} = 12$ and $C_{[3],1} = 18$, respectively. On the second machine, the starting and completion times are calculated as $S_{[3],2} = \max(C_{[2],2}$ (17), $C_{[3],1}$ (18)) = 18 and $C_{[3],2} = 27$, respectively. Before scheduling job 3 on the second machine, the total idle time is calculated as a = max($C_{[3],1}$ (18) - $C_{[2],2}$ (17), 0) = 1. This means that if the production of all three jobs is firmly decided, the no-idle constraint requires the first two jobs on the second machine to start production one unit of time later. On the third machine, the starting and completion times are computed as $S_{[3],3} = \max(C_{[2],3}$ (24) + a (1), $C_{[3],2}$ (27)) = 27 and $C_{[3],3} = 34$, respectively. Before scheduling job 3 on the third machine, the total idle time is calculated as a = a (1) + max ($C_{[3],2}$ (27) – ($C_{[2],3}$ (24)+ a (1)), 0) = 3. This delay consists of 1 unit of time caused by the delay on the second machine and 2 units of time caused by the delay on the third machine. In other words, to satisfy the no-idle constraint, job 1 must be delayed by 1 unit of time on the second machine and 3 units of time on the third machine. The backward computation follows a similar logic using the reversibility property of the problem. In the backward computation, jobs are scheduled in reverse order, from the last job to the first job in the sequence, starting from the last machine. For example, in the given case, job 3 is first scheduled on machine 3, followed by machine 2 and then machine 1. The results of forward and backward computations for this example problem are presented in the matrices below and illustrated in Fig 2.

$$\mathbf{S}_{[\mathbf{k}],\mathbf{i}} = \begin{bmatrix} 0 & 5 & 13 \\ 5 & 13 & 19 \\ 12 & 18 & 27 \end{bmatrix} \mathbf{Sb}_{[\mathbf{k}],\mathbf{i}} = \begin{bmatrix} 29 & 20 & 12 \\ 22 & 16 & 7 \\ 16 & 7 & 0 \end{bmatrix}$$

$$\mathbf{C}_{[\mathbf{k}],\mathbf{i}} = \begin{bmatrix} 5 & 13 & 19 \\ 12 & 17 & 24 \\ 18 & 27 & 34 \end{bmatrix} \mathbf{Cb}_{[\mathbf{k}],\mathbf{i}} = \begin{bmatrix} 34 & 28 & 18 \\ 29 & 20 & 12 \\ 22 & 16 & 7 \end{bmatrix}$$

The acceleration method involves evaluating all permutations created by inserting a single job into every possible position within a given job sequence π. Let $\pi = \{\pi_{(1)}, \pi_{(2)}, \ldots, \pi_{(N-1)}\}$ represent a sequence of N-1 jobs. The goal is to insert job j into each possible position of π, resulting in N complete permutations. Suppose that after N jobs are scheduled to machine i, the start and finish times in the forward (backward) direction are denoted by MS_i (MSb_i) and MC_i (MCb_i), respectively. The process of evaluating a job sequence π, which consists of $N-1$, by inserting an additional job j into each of the N potential

Algorithm 6
Calculation of Cmax in backward direction.

1: $Sb_{[N],M} = 0$ // The job occupying the final position on the last machine
2: $Cb_{[N],M} = Sb_{[N],M} + P_{[N],M}$ // The job occupying the final position on the last machine
3: for ($i \leftarrow M$-1; $i \geq 1$) // The job occupying the final position on the remaining machines
4: $Sb_{[N],i} \leftarrow Cb_{[N],i+1}$
5: $Cb_{[N],i} \leftarrow Sb_{[N],i} + P_{[N],i}$
6: endfor
7: for ($k \leftarrow N$-1; $k \geq 1$) //Jobs placed in positions excluding position N
8: $Sb_{[k],M} \leftarrow Cb_{[k+1],M}$ //last machine
9: $Cb_{[k],M} \leftarrow Sb_{[k],M} + P_{[k],M}$ //machine machine
10: $Sb_{[k],M\text{-}1} \leftarrow$ **max** ($Cb_{[k+1],M\text{-}1}$, $Cb_{[k],M}$) //second last machine
11: $Cb_{[k],M\text{-}1} \leftarrow Sb_{[k],M\text{-}1} + P_{[k],M\text{-}1}$ //second last machine
12: a ← 0
13: a ← **max** ($Cb_{[k],M} - Cb_{[k+1],M\text{-}1}$, 0)
14: for ($i \leftarrow M$-2; $i \geq 1$) //remaining machines
15: $Sb_{[k],i} \leftarrow$ **max** ($Cb_{[k+1],i}$ + a, $Cb_{[k],i+1}$)
16: $Cb_{[k],i} \leftarrow Sb_{[k],i} + P_{[k],i}$
17: a ← a + **max** ($Cb_{[k],i+1} - (Cb_{[k+1],i}$ + a), 0);
18: endfor
19: endfor
20: Return $C_{max} \leftarrow C_{[1],1}$

positions of that sequence using the acceleration method is outlined in Algorithm 7. By executing Algorithm 7, the best position for job j within the job sequence π is determined, along with the minimum completion time for the sequence that includes job j. As can be seen from Algorithm 7, when testing the job for position 1, the relevant job is inserted to the schedule in a backward direction (lines 5-21). Conversely, when testing for the last position, the job is incorporated into the schedule in a forward direction (lines 49-65). For all other positions, excluding the first and last, the job undergoes forward calculations up to the tested position, which are then integrated with the backward calculations of the other remaining jobs in the sequence (lines 22-48) Algorithm 7.

In addition to the ongoing example above, let us consider a fourth job. Assume that the processing times of this new job on machines 1, 2, and 3 are 8, 6, and 4 time units, respectively. Furthermore, let us assume that the four jobs are tested in the sequence π = {1, 2, 4, 3}, where job 4 is inserted into the third position of the original sequence π = {1, 2, 3}. In this case, the acceleration method provided in Algorithm 7 will execute the operations between lines 22 and 48. The matrices $S_{[k],i}$, $Sb_{[k],i}$, $C_{[k],i}$ and $Cb_{[k],i}$ have already been computed. Forward computation is performed for π = {1, 2, 4} as follows:

$MS_1 = C_{[2],1} = 12$ //Starting time on the first machine

$MC_1 = MS_1 + P_{4,1} = 12 + 8 = 20$ //Completion time on the first machine

$MS_2 = \max(C_{[2],2}, MC_1) = \max(17, 20) = 20$ //Starting time on the second machine

$MC_2 = MS_2 + P_{4,2} = 20 + 6 = 26$ //Completion time on the second machine a = $\max(MC_1 - C_{[2],2}, 0) = \max(20 - 17, 0) = 3$ //Idle time on the second machine

$MS_3 = \max(C_{[2],3} + a, MC_2) = \max(24 + 3, 26) = 27$ //Starting time on the third machine

$MC_3 = MS_3 + P_{4,3} = 27 + 4 = 31$//Completion time on the third machine a = a + $\max(MC_2 - (C_{[2],3} + a), 0) = 3 + \max(26 - (24 + 3), 0)$ // Idle time on the third machine

Backward Computation is performed to integrate job 3 into π = {1, 2, 4, 3} as follows:

$L = 0$ and a = 0

$Dummy_C_{max} = MC_1 + Cb_{[3],1} = 20 + 22 = 42$ //Completing time of the first machine after inserting job 3 in backward direction

$L = MC_2 + Cb_{[3],2} = 26 + 16 = 42$ //Completing time of the second machine after inserting job 3 in backward direction a = $\max(Dummy_C_{max} - L, 0) = \max(42 - 42, 0) = 0$ //Idle time for the second machine

$Dummy_C_{max} = \max(Dummy_C_{max}, L) = \max(42, 42) = 42$ //Updating the $Dummy_C_{max}$ after the second machine calculations

$L = MC_3 + a + Cb_{[3],3} = 31 + 0 + 7 = 38$ //Completing time of the third machine after inserting job 3 in backward direction a = a + $\max(Dummy_C_{max} - L, 0) = 0 + \max(42 - 38, 0) = 4$ //Idle time for the third machine

$Dummy_C_{max} = \max(Dummy_C_{max}, L) = \max(42, 38) = 42$//Updating the $Dummy_C_{max}$ after the third machine calculations

$C_{max} = Dummy_C_{max} = 42$

The conventional method evaluates all possible insertion positions for a new job and fully recalculates the makespan for each case. This results in a time complexity of $O(N^2M)$, where N is the number of jobs and M is the number of machines. The proposed acceleration method significantly improves efficiency by utilizing forward and backward scheduling evaluations, which update only the necessary computations. This reduces the overall complexity to $O(NM)$, making the method much more scalable for large problem instances. We validated these theoretical findings through computational experiments. For a test case with 100 jobs and 20 machines, the acceleration method reduced computation time from 23.5 seconds to 4.1 seconds, achieving an 82.6% improvement. These results confirm that the proposed method effectively optimizes large-scale scheduling problems with substantial speed gains.

To evaluate the efficiency of the proposed GRASP algorithm, we analyze its computational complexity across its key components: the construction phase, local search phase, and the destruction-reconstruction mechanism.

The construction phase iterates through each job and evaluates its placement across all factories and machines, calculating the completion time for each possible position. Since each factory has M parallel

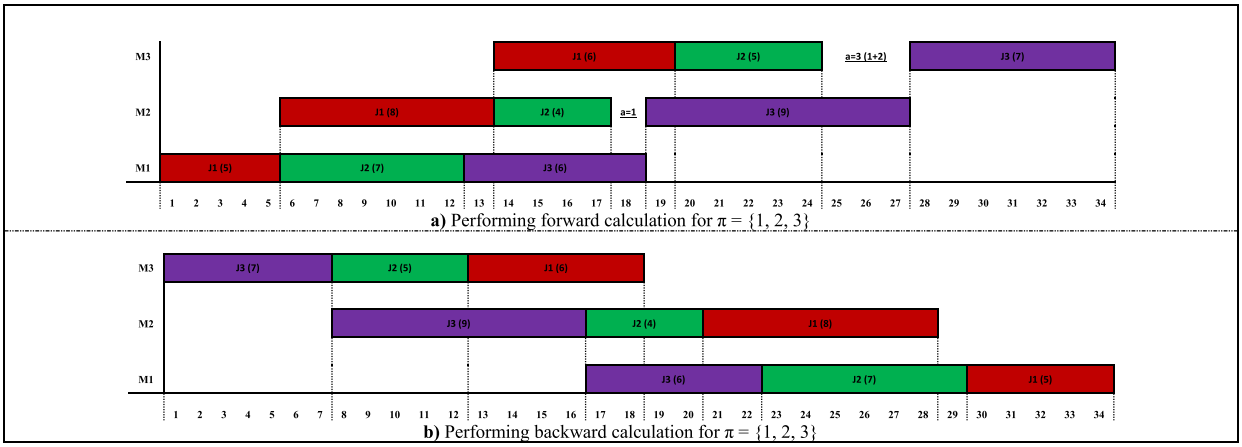


Fig 2. Makespan calculation for the example problem.

Algorithm 7
Acceleration method.

1: Calculate $S_{[k],i}$, $Sb_{[k],i}$, $C_{[k],i}$ and $Cb_{[k],i}$ ($k \leftarrow 1, \ldots, N$-1 and $i \leftarrow 1, \ldots, M$) for job sequence π
2: $Acc_C_{max} \leftarrow$ a big value
3: $p \leftarrow$ -1
4: for ($k \leftarrow 1$; $k \leq N$) //inserting job j into N different positions
5:if (k == 1) //backward calculation
6:$MSb_M \leftarrow Cb_{[k],M}$ //last machine
7:$MCb_M \leftarrow MSb_M + P_{j,M}$ //last machine
8:$MSb_{M-1} \leftarrow$ max ($Cb_{[k],M-1}$, MCb_M) //second last machine
9:$MCb_{M-1} \leftarrow MSb_{M-1} + P_{j,M-1}$ //second last machine
10:a $\leftarrow$ 0
11:a $\leftarrow$ **max(**MCb_M - $Cb_{[k],M-1}$, 0)
12:for (i $\leftarrow$ M - 2; i $\geq$ 1) //rest machines
13:$MSb_i \leftarrow$ max ($Cb_{[k],i}$ + a, MCb_{i+1})
14:$MCb_i \leftarrow MSb_i + P_{j,i}$
15:a $\leftarrow$ a + **max (**MCb_{i+1} - ($Cb_{[k],i}$ + a), 0)
16:endfor
17:$Dummy_C_{max} \leftarrow MCb_1$
18:if ($Acc_C_{max} > Dummy_C_{max}$)
19:$Acc_C_{max} \leftarrow Dummy_C_{max}$
20:$p \leftarrow 1$
21:endif
22:else if (k>1 && k<M)
23:$MS_1 \leftarrow C_{[k-1],1}$ //first machine
24:$MC_1 \leftarrow MS_1 + P_{j,1}$ // first machine
25:$MS_2 \leftarrow$ **max(**$C_{[k-1],2}$, MC_1) //second machine
26:$MC_2 \leftarrow MS_2 + P_{j,2}$ //second machine
27:a $\leftarrow$ 0
28:a $\leftarrow$ **max(**MC_1 - $C_{[k-1],2}$, 0)
29:for ($i \leftarrow 3$; $i \leq M$) //rest machines
30:$MS_i \leftarrow$ max($C_{[k-1],i}$ + a, MC_{i-1})
31:$MC_i \leftarrow MS_i + P_{j,i}$
32:a $\leftarrow$ a + **max(**MC_{i-1} – ($C_{[k-1],i}$ + a), 0)
33:endfor
34:$L \leftarrow$ 0
35:a $\leftarrow$ 0
36:$Dummy_C_{max} \leftarrow MC_1 + Cb_{[k],1}$
37:$L \leftarrow MC_2 + Cb_{[k],2}$
38:a = **max(**$Dummy_C_{max}$ - L, 0)
39:$Dummy_C_{max} \leftarrow$ **max(**$Dummy_C_{max}$, L)
40:for ($i \leftarrow 3$; i $\leq M$)
41:$L \leftarrow MC_i$ + a + $Cb_{[k],i}$
42:a $\leftarrow$ a + **max(**$Dummy_C_{max}$ - L, 0)
43:$Dummy_C_{max} \leftarrow$ **max(**$Dummy_C_{max}$, L)
44:endfor
45:if ($Acc_C_{max} > Dummy_C_{max}$)
46:$Acc_C_{max} \leftarrow Dummy_C_{max}$
47:$p \leftarrow k$
48:endif
49:else //k == M //forward calculation
50:$MS_1 \leftarrow C_{[k-1],1}$ //first machine
51:$MC_1 \leftarrow MS_1 + P_{j,1}$ // first machine
52:$MS_2 \leftarrow$ max ($C_{[k-1],2}$, MC_1) //second machine
53:$MC_2 \leftarrow MS_2 + P_{j,2}$ //second machine
54:a $\leftarrow$ 0;
55:a $\leftarrow$ **max(**MC_1 - $C_{[k-1],2}$, 0)
56:for ($i \leftarrow 3$; $i \leq M$) //rest machines
57:$MS_i \leftarrow$ **max(**$C_{[k-1],i}$ + a, MC_{i-1})
58:$MC_i \leftarrow MS_i + P_{j,i}$
59:a $\leftarrow$ a + **max(**MC_{i-1} – ($C_{[k-1],i}$ + a), 0)
60:endfor
61:$Dummy_C_{max} \leftarrow MC_M$
62:if ($Acc_C_{max} > Dummy_C_{max}$)
63:$Acc_C_{max} \leftarrow Dummy_C_{max}$
64:$p \leftarrow M$
65:endif
66:endif
67: endfor
68: Return $C_{max} \leftarrow Acc_C_{max}$, and the new job sequence after inserting job j into the position p of the sequence π

machines, the worst-case scenario involves checking all N jobs across F factories, and for each factory, distributing the jobs among M machines. Thus, the complexity of this step is O(NFM). Additionally, forming the Restricted Candidate List (RCL) and selecting a job from the list incurs an additional complexity of O(N log N). Since this phase is repeated for each job, the overall complexity of the construction phase remains O(NFM).

The local search phase applies a modified LS3 algorithm that selects a factory based on roulette wheel selection and then evaluates alternative job sequences to minimize the makespan. The worst-case complexity arises from testing all N jobs across all F factories and distributing them across M machines, leading to O(NFM), while evaluating all neighboring solutions results in a complexity of O(N^2FM). Therefore, the overall local search complexity is approximated as O(N^2FM).

The destruction-reconstruction mechanism removes a subset of jobs and reinserts them using the construction phase approach. The destruction step itself incurs O(d) operations, while the reconstruction phase follows the same complexity as the initial construction phase, O(NFM). Hence, the overall complexity of this step is O(NFM).

Since GRASP is an iterative metaheuristic, where the above steps are repeated over iterations, the total complexity of the algorithm can be expressed as: $O\left(Grasp_Max_Step\left(NFM + N^2FM + NFM\right)\right) = O\left(Grasp_Max_StepN^2FM\right)$

This analysis shows that the computational complexity of GRASP is polynomial in nature, making it feasible for large-scale scheduling problems compared to exact methods, which typically exhibit exponential complexity.

7. Computational study

This section is organized into three parts. The first part focuses on tuning the parameters for the GRASP algorithm introduced in this study. In the second part, the performance of the proposed mathematical models is evaluated and compared with other exact solution techniques. The third part presents a computational analysis to assess how well the GRASP algorithm performs. All algorithms and models are implemented using Visual C++, with the CPLEX 12.7.1 solver utilized to solve the mathematical models and exact solution methods in the Visual C++ environment. To enhance reproducibility and support further research, the full implementation of all algorithms and models proposed in this paper is available at: https://github.com/alperhamzadayi/dist-noidle-fsp. The experiments are conducted on a machine with an Intel Core i5-2450M processor (2.5 GHz) and 4 GB of RAM. The models and algorithms are tested using DPFSP instances provided by Naderi and Ruiz [10], which are accessible at http://soa.iti.es, considering the no-idle restriction for jobs. The problem instances are categorized into two groups: small and large. The small-instance set includes various combinations of jobs (N = {4, 6, 8, 10, 12, 14, 16}), factories (F = {2, 3, 4}), and machines (M = {2, 3, 4, 5}). For each combination of N, F, and M, five instances are generated, totaling 420 small-sized problem instances. On the other hand, the large-instance set is based on benchmark instances introduced by Taillard [67] for multiple factories. This set covers 12 combinations of N and M. For machines M = 5, 10, and 20, the corresponding values for N are {20, 50, 100}, {20, 50, 100, 200}, and {20, 50, 100, 200, 500}, respectively, with 10 instances generated for each combination. Naderi and Ruiz [10] later adapted these instances

Table 2
Parameter calibration results for small instances.

Experiment No	α	d	Avg. RPD	RBS
S1	0.4	1	0.4445	51
S2	0.5	1	0.3495	61
S3	0.6	1	0.4407	55
S4	0.4	2	0.1259	76
S5	0.5	2	0.1218	78
S6	0.6	2	0.2235	70
S7	0.4	3	0.2256	68
S8	0.5	3	0.3236	62
S9	0.6	3	0.2238	65

Table 3
Parameter calibration results for large instances.

Experiment No	α	*d*	Avg. RPD	RBS
L1	0.4	8	0.4139	24
L2	0.5	8	0.7232	15
L3	0.6	8	0.6258	14
L4	0.4	10	0.8191	17
L5	0.5	10	0.7238	15
L6	0.6	10	1.0310	12
L7	0.4	12	0.4213	20
L8	0.5	12	1.1239	13
L9	0.6	12	1.3343	11

Table 4
Overview of the comparisons of the mathematical models.

Model	#Optimal	#Suboptimal	Gap%	Time (Seconds)
PB (Ying et al. [9])	253	167	9.447	215.001
SB	265	155	5.277	200.696
IV_PB	346	74	2.599	116.950
IV_PB + SBC1	351	69	2.418	112.705
IV_PB + SBC2	357	63	2.149	106.887
IV_PB + SBC3	363	57	1.616	90.094
IV_PB + SBC4	366	54	1.556	87.381

for DPFSP by adding factory F = {2, 3, 4, 5, 6, 7}, resulting in 720 large-sized problem instances. The processing times for both sets are uniformly distributed between 1 and 99.

7.1. Parameter calibration of GRASP

GRASP requires calibration of three distinct parameters: the maximum number of global iterations (*Grasp_Max_Step*), the alpha parameter (α), and the number of jobs to be removed from a given solution during the destruction phase (d). Actually, there is no need to calibrate *Grasp_Max_Step* since the algorithm is run within a specified CPU time limit as in the literature. For the parameter calibration tests, the CPU time limit is set to $N \times M \times 30$ milliseconds.

Parameter calibration relies on the average relative percentage deviation (RPD), which is computed as $\text{RDP} = (\text{Some}^{\text{pc}} - \text{Best}_{\text{sol}})\backslash\text{Best}_{\text{sol}}$, where Some^{pc} represents the best makespan achieved among the independent runs by the parameter combination pc, and Best_{sol} is the best makespan value among all compared parameter combinations. If several parameter combinations result in the smallest average RPD, the one with the highest count of relatively better solutions (RBS) is selected as the alternative measure.

The parameter calibration experiment for small-size instances is conducted on the 84 instances (using the first combination of the small-size instances). The levels of the parameters, α and *d*, are determined through preliminary experiments and are varied at three levels: {0.4, 0.5, 0.6} and {1, 2, 3}, respectively. For each parameter combination, the algorithm is run 5 times (replications). In total, the algorithm is run 3780 times (84 × 3 × 3 × 5). The results of the parameter calibration experiment for small problem instances are presented in Table 2. It is evident that the parameter combination S5 ($\alpha = 0.5$, $d = 2$) yields the lowest average RPD of 0.07, along with a higher number of relatively better solutions (RBS = 78).

The parameter calibration experiment for the large problem instances is carried out using 48 randomly selected benchmark instances. Similarly, the levels of the parameters, α and *d*, are determined through preliminary experiments and are varied at three levels: {0.4, 0.5, 0.6} and {8, 10, 12}, respectively. The algorithm is executed 5 times (replications) for each parameter combination. In total, the algorithm is run 2160 times (48 × 3 × 3 × 5). Table 3 displays the results of the parameter calibration experiment for large problem instances. The parameter combination L1 (α = 0.4, d = 8) clearly demonstrates the lowest average RPD of 0.4139, while also achieving the highest RBS value of 24.

Table 5
Summarized comparisons of IV_PB + SBC4 and RIV_PB + SBC4 models.

Model	#Optimal	#Suboptimal	Gap%	Time (Seconds)
IV_PB + SBC4	366	54	1.556	87.381
RIV_PB + SBC4	385	35	0.845	57.381

Table 6
Performance comparison of Benders algorithms and RIV_PB + SBC4 model.

Model	#Optimal	#Suboptimal	Gap %	Time (Seconds)	# Cut
RIV_PB + SBC4	385	35	0.845	57.381	-
ABD_RIV_PB + SBC4	380	40	1.256	188.289	45.431
BD_RIV_PB + SBC4	394	26	0.463	171.810	31.056
HBD_RIV_PB + SBC4	419	1	0.011	147.996	15.960

7.2. Performance evaluation of mathematical models and other exact solution methods

This section of the experiments compares eleven exact methods. These include two models introduced in Section 2: PB and SB, as well as the IV_PB model from Section 2. Additionally, four variations of the IV_PB model are considered, which are derived by applying different symmetry-breaking constraints (represented as IV_PB + SBC1, IV_PB_SBC2, IV_PB_SBC3, and IV_PB_SBC4). The model generated by restricting job positions in the IV_PB model (denoted as RIV_PB), as discussed in Sections 3 and 4, is also compared. Additionally, the version 12.7.1 of CPLEX's automatic Benders decomposition algorithm, applied to the RIV_PB model (referred to as ABD_RIV_PB), is considered in the comparison. The comparison also includes the pure and hybrid Benders decomposition algorithms proposed in Section 5, which are represented as BD_RIV_PB and HBD_RIV_PB, respectively. First, we carry out a comparative analysis among the position-based (PB) model proposed by Ying et al. [9], the proposed sequence based (SB) model, the proposed improved position-based (IV_PB) model and IV_PB model with four different symmetry constraints described in Section 2.2. Model tests are conducted using 420 small-size instances, and each model is solved within a time limit of 500 seconds. During the execution of each model, the activation of the CPLEX presolver is managed automatically by the solver.

In each model run, the CPLEX solver operates in deterministic mode using four threads. The summarized results of the tests are presented in Table 4. The first column of Table 4 shows each model, with the second and third columns representing the number of instances solved optimally (#Optimal) and those that are not guaranteed to be solved optimally within the specified time limit (#Suboptimal). The "Gap" column displays the final optimality gap in percentage, recorded at the end of the optimization process, whether the optimal solution is found or the time limit is reached. Finally, the "Time" column displays the average computation time for the 420 instances that are analyzed.

As shown in Table 4, the IV_PB + SBC4 model solves 366 out of 420 instances optimally, with an average gap value of 1.556%. Following the IV_PB + SBC4 model, the rankings are as follows: IV_PB + SBC3 with 363 optimal solutions, IV_PB + SBC2 with 357, IV_PB + SBC1 with 351, IV_PB with 346, SB with 265, and PB with 253 optimal solutions. Their respective average gap values, in the same order, are 1.616%, 2.149%, 2.418%, 2.599%, 5.277%, and 9.447%. The IV_PB + SBC4 model solves the instances in the shortest average time of 87.381 seconds. It is followed by the IV_PB + SBC3 with an average of 90.094 seconds, IV_PB + SBC2 at 106.887 seconds, IV_PB + SBC1 at 112.705 seconds, IV_PB at

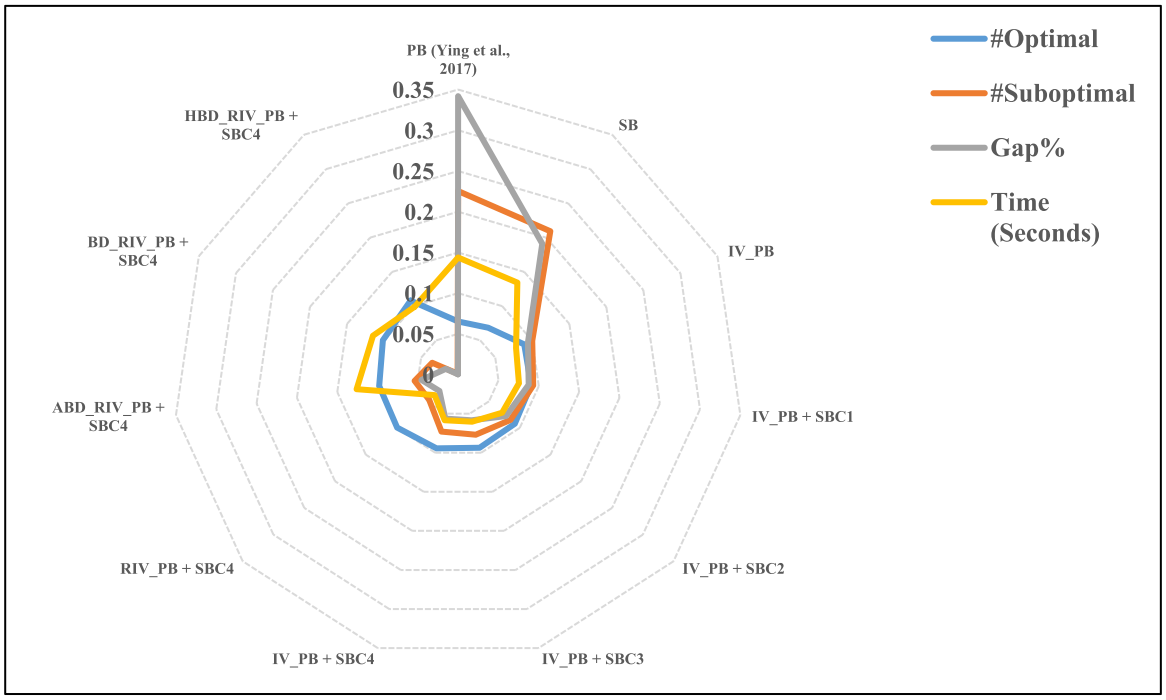


Fig 3. Results of all performance measures for all compared exact methods.

116.950 seconds, SB at 200.696 seconds, and finally the PB model at 215.001 seconds. It is clear that the proposed IV_PB model, along with the SBC4 symmetry-breaking constraint, demonstrates the best performance among seven compared models by optimally solving the most instances with the shortest average time and the lowest average gap value. In a similar manner, the PB model introduced by Ying et al. [9] showed the worst performance among the seven models across all performance measures evaluated.

In this section, the second test involves comparing the performance of the RIV_PB model, which is generated by limiting the number of positions in the IV_PB model, with the IV_PB model itself. As demonstrated by the tests conducted above, the SBC4 symmetry-breaking constraint yields the best performance when combined with the IV_PB model. Therefore, tests are performed by inserting the SBC4 to RIV_PB model. The SBC4 constraint is inserted to the RIV_PB model by restricting the number of positions to Z, as can be seen in Constraint set (62) below.

$$\sum_{j=1}^{N}\sum_{k=1}^{Z} 2^{N-j} \times X_{j,k,f} \geq \sum_{j=1}^{N}\sum_{k=1}^{Z} 2^{N-j} \times X_{j,k,f+1} \quad f \in \{1, \ldots, F-1\} \tag{62}$$

For each instance, to determine an upper limit on the number of

Table 7
The average ARPD and RBS values for the algorithms being compared, based on small problem instances.

(F, N, M)	HBD_RIV_PB + SBC4		IRG		GRASP_WLS		GRASP	
	RDP	RBS	Avg. ARDP	RBS	Avg. ARDP	RBS	Avg. ARDP	RBS
(2, 4, M)	0.0000	20	0.0000	20	0.0000	20	0.0000	20
(2, 6, M)	0.0000	20	0.0000	20	0.0001	20	0.0000	20
(2, 8, M)	0.0000	20	0.0000	20	0.0003	20	0.0000	20
(2, 10, M)	0.0000	20	0.0003	20	0.0013	20	0.0007	20
(2, 12, M)	0.0000	20	0.0032	16	0.0039	18	0.0020	18
(2, 14, M)	0.0000	20	0.0040	16	0.0076	13	0.0036	17
(2, 16, M)	0.0000	20	0.0072	11	0.0096	10	0.0054	11
Average	0.0000	20 (140)	0.0021	17.57 (123)	0.0033	17.28 (121)	0.0017	18 (126)
(3, 4, M)	0.0000	20	0.0000	20	0.0000	20	0.0000	20
(3, 6, M)	0.0000	20	0.0000	20	0.0000	20	0.0000	20
(3, 8, M)	0.0000	20	0.0019	20	0.0010	20	0.0002	20
(3, 10, M)	0.0000	20	0.0018	20	0.0034	19	0.0022	20
(3, 12, M)	0.0000	20	0.0066	16	0.0090	17	0.0033	18
(3, 14, M)	0.0000	20	0.0107	11	0.0132	7	0.0068	17
(3, 16, M)	0.0001	19	0.0153	5	0.0160	4	0.0097	10
Average	0.0000	19.85 (139)	0.0052	16 (112)	0.0061	15.28 (107)	0.0032	17.85 (125)
(4, 4, M)	0.0000	20	0.0000	20	0.0000	20	0.0000	20
(4, 6, M)	0.0000	20	0.0000	20	0.0000	20	0.0000	20
(4, 8, M)	0.0000	20	0.0000	20	0.0000	20	0.0000	20
(4, 10, M)	0.0000	20	0.0006	20	0.0013	20	0.0014	20
(4, 12, M)	0.0000	20	0.0031	18	0.0075	15	0.0021	18
(4, 14, M)	0.0000	20	0.0070	20	0.0135	13	0.0060	18
(4, 16, M)	0.0000	20	0.0159	4	0.0178	6	0.0119	7
Average	0.0000	20 (140)	0.0038	17.42 (122)	0.0057	16.28 (114)	0.0031	17.57 (123)
All Avg.	0.0000	19.95 (419)	0.0037	17.00 (357)	0.0050	16.28 (342)	0.0026	17.80 (374)

Table 8
Paired t-tests on the average ARPD between GRASP_WLS and IRG.

N	Mean	S-Dev	t-value	P-value	Significant
4	0.000000	0.0000	-	-	No
6	0.000028	0.0002	1.00	0.321	No
8	-0.000200	0.0033	-0.47	0.640	No
10	0.001112	0.0028	3.01	0.004	Yes
12	0.002488	0.0059	3.22	0.002	Yes
14	0.004261	0.0054	5.97	0.000	Yes
16	0.001617	0.0037	3.36	0.001	Yes

Table 9
Paired t-tests on the average ARPD between GRASP and IRG.

N	Mean	S-Dev	t-value	P-value	Significant
4	0.000000	0.0000	-	-	No
6	0.000000	0.0000	-	-	No
8	0.000552	0.0042	1.00	0.321	No
10	-0.000541	0.0027	-1.55	0.127	No
12	0.001844	0.0046	3.09	0.003	Yes
14	0.001867	0.0044	3.19	0.002	Yes
16	0.003653	0.0044	6.48	0.000	Yes

Table 10
Average rankings achieved by Friedman test on small instances.

Algorithms	Rankings
GRASP	4.3309
IRG	4.9797
GRASP_WLS	5.6892

positions, the procedure described in Section 3 is followed, and the results are provided in Appendix. Here, the *h_Cmax* value used to calculate the *Z* value via the mathematical model described in Section 3 is obtained by running GRASP for $N \times M \times 20$ milliseconds. Assuming that the times spent to find the *h_Cmax* and *Z* values are t_h and t_m respectively. The RIV_PB + SBC4 model is run for 500 – (t_h + t_m) seconds to make a fair comparison. In other words, the time spent to find the *Z* value and the running time of the model are provided to be 500 seconds in total. All instances are optimally solved in a very short time using the mathematical model developed to determine the *Z* value. While solving 420 instances, this model solves the most time-consuming instance in 8.768 seconds, the least time-consuming instance in 0.007 seconds, and the average time for all instances is 0.218 seconds. The summarized comparison of IV_PB + SBC4 and RIV_PB + SBC4 are shown in Table 5.

As illustrated in Table 5, although the RIV_PB + SBC4 model required extra time to calculate the *Z* value prior to execution, it still achieved more optimal solutions (385 optimal solutions) with a lower average gap value (0.845%) and a shorter average computational time (57.381 seconds) compared to the IV_PB + SBC4 model.

For the final test in this section, the results of the RIV_PB + SBC4 model are compared to those obtained from applying the automatic Benders decomposition algorithm in CPLEX version 12.7.1 to the RIV_PB + SBC4 model (referred to as ABD_RIV_PB + SBC4). Additionally, comparisons are made with both the pure and hybrid Benders decomposition methods presented in Section 5 (referred to as BD_RIV_PB + SBC4 and HBD_RIV_PB + SBC4, respectively). For a fair comparison, the same *Z* values obtained in the previous test are used, and, similarly, the same (t_h + t_m) time values obtained in the previous test are subtracted from 500 seconds to determine the remaining time for running ABD_RIV_PB + SBC4, BD_RIV_PB + SBC4, and HBD_RIV_PB + SBC4. Additionally, in each iteration of HBD_RIV_PB, GRASP is executed with a time limit of $N \times M \times 10$ milliseconds, and the running times of GRASP in each iteration are included in the total running time of HBD_RIV_PB + SBC4. Four threads are used in the CPLEX solver running in deterministic mode for each model execution. In the implementation of BD_RIV_PB + SBC4 and HBD_RIV_PB + SBC4, the CPLEX presolver is disabled, whereas for the other models, the decision is made by the CPLEX solver. The performance comparison of Benders Algorithms and RIV_PB + SBC4 model is presented in Table 6. In Table 6, the #Cuts column shows the average number of Benders optimality cuts utilized during the optimization process.

As can be seen from Table 6, HBD_RIV_PB + SBC4 successfully solves 419 instances optimally. It is followed by BD_RIV_PB + SBC4, ABD_RIV_PB + SBC4, and RIV_PB + SBC4, which achieve 394, 380, and 385 optimal solutions, respectively. RIV_PB + SBC4 achieves the shortest average solution time of 57.381 seconds for the instances. Following this, HBD_RIV_PB + SBC4, BD_RIV_PB + SBC4, and ABD_RIV_PB + SBC4 have average times of 147.996, 171.810, and 188.289 seconds, respectively. It is clear that HBD_RIV_PB + SBC4 exhibits the shortest average solution time among the Benders algorithms analyzed in this comparison. In Benders decomposition algorithm, the master problem typically demands the most time to solve, whereas the submodels are solved rapidly during the iterations, since they are linear programming models. Therefore, the time required for the submodels is insignificant in comparison to that for the master models. In terms of average solution gap, HBD_RIV_PB + SBC4 reaches the lowest average gap value with 0.011. It is followed by the BD_RIV_PB + SBC4, RIV_PB +

Table 11
The average ARPD and RBS of IRG and GRASP for large instances with q = 30.

(F, N, M)	IRG		GRASP		(F, N, M)	IRG		GRASP	
	RBS	Avg. ARDP	RBS	Avg. ARDP		RBS	Avg. ARDP	RBS	Avg. ARDP
(2, 20, M)	10	0.0140	15	0.0088	**(5, 20, M)**	28	0.0045	26	0.0051
(2, 50, M)	10	0.0124	18	0.0091	**(5, 50, M)**	11	0.0117	14	0.0078
(2, 100, M)	8	0.0157	19	0.0108	**(5, 100, M)**	7	0.0122	13	0.0094
(2, 200, M)	5	0.0138	12	0.0115	**(5, 200, M)**	6	0.0090	7	0.0088
(2, 500, M)	2	0.0077	3	0.0071	**(5, 500, M)**	3	0.0098	3	0.0093
Average	0.2916 (35)	0.0135	0.5583 (67)	0.0097	***Average***	0.4583 (55)	0.0094	0.5250 (63)	0.0078
(3, 20, M)	10	0.0130	18	0.0119	**(6, 20, M)**	30	0.0003	30	0.0001
(3, 50, M)	11	0.0122	15	0.0090	**(6, 50, M)**	13	0.0091	12	0.0080
(3, 100, M)	11	0.0134	17	0.0122	**(6, 100, M)**	11	0.0114	15	0.0092
(3, 200, M)	4	0.0107	10	0.0098	**(6, 200, M)**	7	0.0086	8	0.0086
(3, 500, M)	5	0.0080	1	0.0098	**(6, 500, M)**	3	0.0080	4	0.0081
Average	0.3416 (41)	0.0121	0.5083 (61)	0.0107	***Average***	0.5333 (64)	0.0073	0.5750 (69)	0.0064
(4, 20, M)	12	0.0160	14	0.0156	**(7, 20, M)**	30	0.0006	30	0.0001
(4, 50, M)	12	0.0116	15	0.0109	**(7, 50, M)**	8	0.0104	19	0.0070
(4, 100, M)	12	0.0119	14	0.0101	**(7, 100, M)**	5	0.0127	18	0.0087
(4, 200, M)	5	0.0135	8	0.0115	**(7, 200, M)**	11	0.0064	11	0.0050
(4, 500, M)	2	0.0056	2	0.0064	**(7, 500, M)**	4	0.0078	4	0.0088
Average	0.3583 (43)	0.0126	0.4416 (53)	0.0116	***Average***	0.4833 (58)	0.0076	0.6833 (82)	0.0055
All Average						0.3486 (251)	0.0126	0.5486 (395)	0.0086

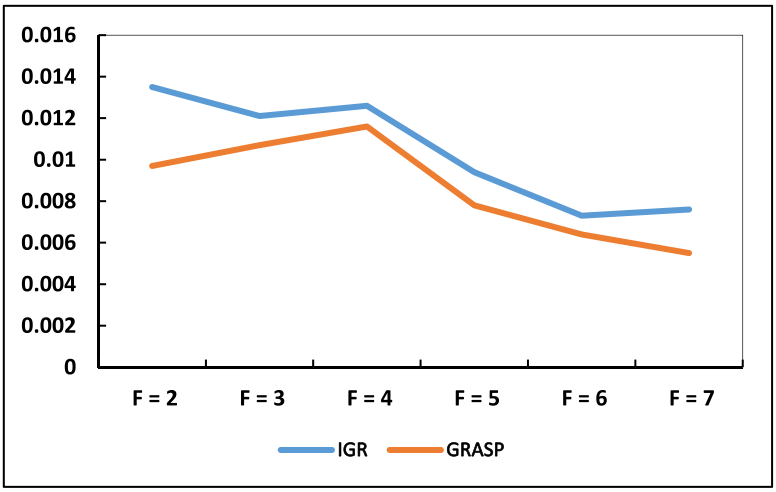


Fig 4. Average ARDP values of compared algorithms for q = 30.

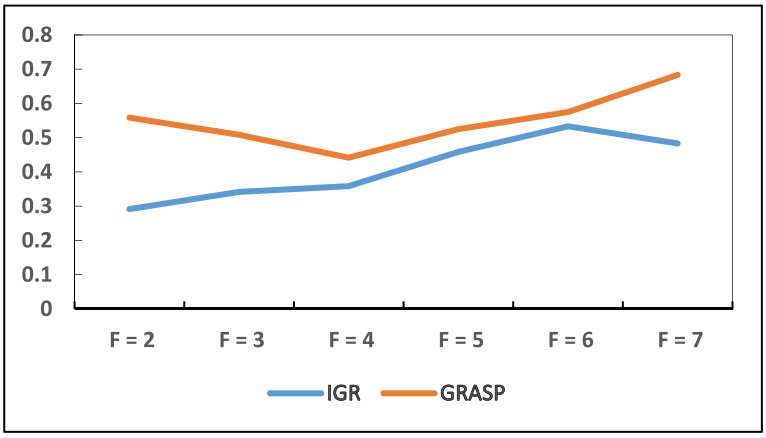


Fig 5. Average RBS values of compared algorithms for q = 30.

SBC4 and ABD_RIV_PB + SBC4 with the gap averages of 0.463%, 0.845% and 1.256% respectively. When examining the iterations of HBD_RIV_PB + SBC4, BD_RIV_PB + SBC4, and ABD_RIV_PB + SBC4, it is apparent that the Benders iterations are significantly reduced by adding an extra cut, generated via a heuristic, to the master model in each iteration. To facilitate the analysis of experimental results, the results of all performance measurements for all compared exact methods are normalized between 0 and 1 and are shown in Fig 3. As can be seen from Fig 3, HBD_RIV_PB + SBC4 outperforms the other exact solution methods by optimally solving 419 instances, while the PB model shows the worst performance, achieving optimal solutions for only 253 instances, confirming what is declared by the authors in their paper.

It should be noted that the HBD_RIV_PB + SBC4 model could not optimally solve only the instance named I_3_16_5_5, with a gap value of 4.83%, out of 420 instances. In Appendix, the solution values obtained for each instance using the HBD_RIV_PB + SBC4 model, the number of cuts applied during the solution process, and the solution times for each instance are provided in the columns labeled O_Value, #Cuts, and S_Time, respectively.

7.3. Computational tests for GRASP

In this subsection, the performance of GRASP is first compared with the HBD_RIV_PB + SBC4 model using small-size instances, as well as the Iterated Reference Greedy algorithm (IRG) proposed by Ying et al. [9], which is the only approximation algortihm developed in the literature for solving $DF_m|prmu;\ no_idle|C_{max}$. In their study, Ying et al. [9] adapted the iterated greedy (IG_{PR}) algorithm, originally developed by Pan and Ruiz [26] to solve the mixed no-idle permutation flowshop scheduling problem, to solve $DF_m|prmu;\ no_idle|C_{max}$. The test results from Ying et al. [9] demonstrated that IRG outperforms the IG_{PR} algorithm. For this purpose, IRG is reimplemented in this paper. Additionally, to better analyze the fundamental performance of GRASP, the case of GRASP without the local search algorithm (GRASP_WLS) used in the experiments (see line 5 of Algorithm 1) are run separately. The CPU time limit for running the approximation algortihms is set to N × M × 50 milliseconds and all approximation algortihms are executed 5 times (replications). The solution results of the HBD_RIV_PB + SBC4 model are taken from the previous subsection. Average Relative Percent Deviation (ARDP) is used as the response variable in algorithm comparisons (see Eq. (63)).

$$\text{ARDP} = \left(\sum_{r=1}^{R} \frac{\text{Some}_{alg}^{r} - \text{Best}_{sol}}{\text{Best}_{sol}} \right) \times \frac{1}{R} \tag{63}$$

where Some_{alg}^{r} is the solution obtained by the replication r of the considered algorithm run on a given instance and Best_{sol} is the optimal or the lowest known upper bound value known for the relative instance. The average ARPD and RBS values for the algorithms compared in this study, based on small problem instances, are presented in Table 7. These values are grouped according to the different values of F and N used in the experiments. As shown in Table 7, when analyzing the lowest average ARPD values for various F, GRASP delivers the best performance among the approximation algorithms. It is followed by IRG and

Table 12
Paired t-tests on the average RPD of GRASP and IRG for large instances for q = 30.

N	Mean	S-Dev	t-value	P-value	Significant
20	0.0011	0.0059	2.52	0.013	Yes
50	0.0026	0.0062	5.61	0.000	Yes
100	0.0028	0.0067	5.68	0.000	Yes
200	0.0011	0.0067	1.88	0.062	No
500	-0.0004	0.0041	-0.81	0.419	No

GRASP_WLS, respectively. Similarly, when considering the average ARPD values across all F, GRASP once again outperforms the other approximation algorithms, achieving a value of 0.0026. IRG and GRASP_WLS follow with values of 0.0037 and 0.0050, respectively. It is important to note that all the compared approximation algorithms achieve ARPD values close to 0 on average, indicating that they are all effective for solving small-sized instances. The average total RBS values are also displayed in Table 7. Among the approximation algorithms, GRASP again yields the highest average total RBS value for all F, with a value of 17.80 (374). It is followed by IRG with 17.00 (357) and GRASP_WLS with 16.28 (342). Based on these results, it is clear that GRASP demonstrates a strong potential for solving small-sized instances, even without the inclusion of a local search algorithm (in its pure form). As highlighted in the previous subsection, the HBD_RIV_PB + SBC4 algorithm successfully solves all instances to optimality, except for the instance labeled I_3_16_5_5. While HBD_RIV_PB + SBC4 solved this instance with the objective function value of 537, the value of 535 is obtained with GRASP for the relevant instance. A paired t-test is conducted between IRG and GRASP_WLS at a 5% confidence level for each distinct N and the test results are given in Table 8. As shown in Table 8, the difference between the means of both algorithms is insignificant for N = 4, 6, and 8, while it becomes significant for N = 10, 12, 14, and 16. Similarly, a paired t-test is conducted between IRG and GRASP and the test results are given in Table 9. In this comparison, the difference between the means of both algorithms is insignificant for N = 4, 6, 8, and 10, whereas it is found to be significant for N = 12, 14, and 16. A Friedman test is also conducted on the small-sized instances to analyze the performance of the algorithms, as suggested by Derrac et al. [68]. The test provides a ranking of the algorithms, with lower values indicating better rankings. The ranking values are presented in Table 10. Based on these results, it can be concluded that GRASP ranks the highest among the three, followed by IRG and GRASP_WLS.

In this paper, the performances of IRG and GRASP are compared using a large set of 720 instances as the second test, under three preset maximum CPU times, i.e., N × M × q, where q = {30, 90}. Algorithms are executed 5 times (replications). Table 11 compares IRG and GRASP across different problem configurations for q = 30. As can be seen from Table 11, GRASP generally performs better in terms of both the number

Table 13
The average ARPD and RBS of IRG and GRASP for large instances with q = 90.

(F, N, M)	IRG		GRASP		(F, N, M)	IRG		GRASP	
	RBS	Avg. ARDP	RBS	Avg. ARDP		RBS	Avg. ARDP	RBS	Avg. ARDP
(2, 20, M)	13	0.0099	17	0.0067	**(5, 20, M)**	27	0.0050	29	0.0036
(2, 50, M)	2	0.0104	29	0.0059	**(5, 50, M)**	14	0.0092	17	0.0068
(2, 100, M)	8	0.0099	23	0.0054	**(5, 100, M)**	13	0.0088	19	0.0061
(2, 200, M)	8	0.0133	12	0.0088	**(5, 200, M)**	10	0.0077	10	0.0082
(2, 500, M)	3	0.0072	7	0.0052	**(5, 500, M)**	7	0.0055	3	0.0085
Average	0.2833 (34)	0.0104	0.7333 (88)	0.0064	***Average***	0.5916 (71)	0.0075	0.6500 (78)	0.0062
(3, 20, M)	17	0.0114	15	0.0106	**(6, 20, M)**	30	0.0001	30	0.0001
(3, 50, M)	17	0.0084	15	0.0067	**(6, 50, M)**	17	0.0085	14	0.0079
(3, 100, M)	15	0.0103	17	0.0089	**(6, 100, M)**	15	0.0080	17	0.0061
(3, 200, M)	12	0.0096	8	0.0094	**(6, 200, M)**	13	0.0076	10	0.0074
(3, 500, M)	6	0.0069	5	0.0085	**(6, 500, M)**	6	0.0069	4	0.0069
Average	0.5583 (67)	0.0097	0.5000 (60)	0.0088	***Average***	0.6750 (81)	0.0060	0.6250 (75)	0.0053
(4, 20, M)	14	0.0164	19	0.0157	**(7, 20, M)**	29	0.0008	30	0.0002
(4, 50, M)	7	0.0092	23	0.0060	**(7, 50, M)**	8	0.0107	23	0.0080
(4, 100, M)	11	0.0114	19	0.0080	**(7, 100, M)**	15	0.0079	18	0.0056
(4, 200, M)	9	0.0086	12	0.0083	**(7, 200, M)**	15	0.0052	9	0.0058
(4, 500, M)	3	0.0067	7	0.0062	**(7, 500, M)**	6	0.0062	4	0.0073
Average	0.3666 (44)	0.0112	0.6666 (80)	0.0093	***Average***	0.6083 (73)	0.0062	0.7000 (84)	0.0050
All Average						0.5138 (370)	0.0085	0.6458 (465)	0.0068

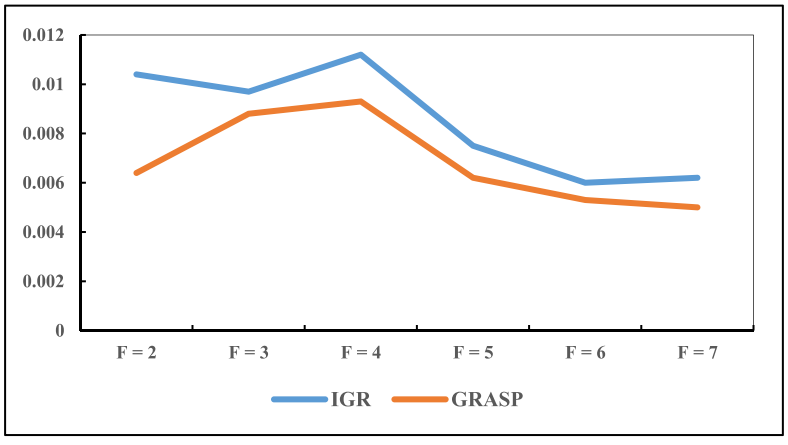


Fig 6. Average ARDP values of compared algorithms for q = 90.

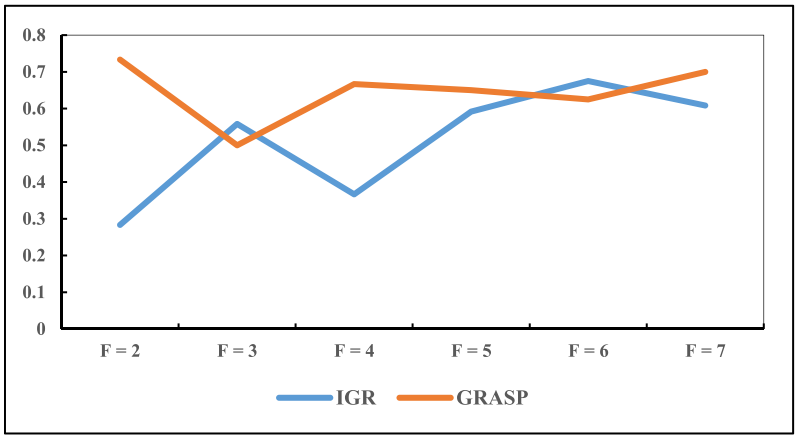


Fig 7. Average RBS values of compared algorithms for q = 90.

Table 14
Paired t-tests on the average RPD of GRASP and IRG for large instances for q = 90.

N	Mean	S-Dev	t-value	P-value	Significant
20	0.0011	0.0060	2.50	0.013	Yes
50	0.0025	0.0049	6.82	0.000	Yes
100	0.0026	0.0054	6.63	0.000	Yes
200	0.0006	0.0066	1.11	0.268	No
500	-0.0005	0.0050	-0.85	0.401	No

of better solutions found and the quality of those solutions. For instance, in the configuration (2, 20, M), GRASP finds 15 better solutions with an Avg. ARDP of 0.0088, whereas IRG finds only 10 better solutions with an Avg. ARDP of 0.0140. Similarly, in the (5, 500, M) configuration, both algorithms find 3 better solutions, but GRASP achieves a slightly better Avg. ARDP of 0.0093 compared to IRG's 0.0098. Overall, GRASP has a higher average RBS of 0.5486 (395 better solutions) compared to IRG's 0.3486 (251 better solutions), and a lower average Avg. ARDP of 0.0086, while IRG has a higher average Avg. ARDP of 0.0126, indicating that GRASP consistently finds more high-quality solutions. This trend holds true across all problem sizes, with GRASP demonstrating superior performance, especially for larger problems.

Figs. 4 and 5 present the average ARPD and RBS results for the compared algorithms when q = 30, respectively.

A paired t-test is conducted between IRG and GRASP at a 5% confidence level for each distinct N when q = 30 and the test results are given in Table 12. For N = 20, the result is statistically significant with a p-value of 0.013. For N = 50 and N = 100, both show highly significant results with p-values of 0.000. However, for N = 200, the result is not significant (p = 0.062), and for N = 500, the result is also not significant with a p-value of 0.419. In general, smaller sample sizes show significant results, while larger sample sizes do not.

Table 13 compares IRG and GRASP across different problem configurations for q = 90. In the configuration (2, 20, M), GRASP performs better than IRG, finding 17 better solutions with an Avg. ARDP of 0.0067, while IRG finds only 13 solutions with an Avg. ARDP of 0.0099. Similarly, for (5, 20, M), GRASP achieves 29 better solutions with an Avg. ARDP of 0.0036, while IRG finds 27 solutions but with a higher Avg. ARDP of 0.0050. In larger problem sizes, such as (5, 500, M), even though IRG finds more solutions (7 vs. 3 for GRASP), GRASP still produces better quality solutions, with an Avg. ARDP of 0.0085 compared to IRG's 0.0055. When looking at the averages across all configurations, GRASP consistently performs better, finding 465 solutions compared to IRG's 370, and having a lower overall Avg. ARDP of 0.0068 versus IRG's 0.0085. GRASP generally performs better by finding more solutions and delivering higher-quality results. The difference in performance is especially noticeable in larger problem sizes, where GRASP shows its strength. IRG, while competitive in smaller problem sizes, falls behind in larger instances.

Figs. 6 and 7 present the average ARPD and RBS results for the compared algorithms when q = 90, respectively.

A paired t-test is performed between IRG and GRASP at a 5% confidence level for each N when q = 90, with the results shown in Table 14. For N = 20, the result is significant with a p-value of 0.013. For N = 50 and N = 100, the results are highly significant with p-values of 0.000. However, for N = 200 and N = 500, the results are not significant, with p-values of 0.268 and 0.401, respectively. In conclusion, significant results are observed for smaller sample sizes, while larger ones show no significance.

8. Conclusions

This paper addresses $DF_m|prmu;\ no_idle|C_{max}$ which is a more general case than both the DPFSP and NIPFSP. Apart from the work by Ying et al. [9], which is the only study that deals with this problem, no other research has been conducted in this area. Ying et al. [9] proposed a position-based MILP model and an IRG to solve the problem. They also adapted IG_{PR} by Pan and Ruiz [26] to evaluate the performance of IRG and compare it with a state-of-the-art algorithms. Their findings showed that IRG outperforms IG_{PR} in terms of average RPD and RBS. To bridge the gap for DNIPFSP, this paper presents three MILP models and GRASP to efficiently solve this problem. Although GRASP is widely used in scheduling literature, it has not been applied to any variant of distributed flow shop scheduling problems before. LS3 algorithm is modified to improve not only the critical factory (with the highest makespan) but also other factories, and this modification is incorporated into GRASP as a local search. The MILP models presented in this paper include a sequence-based (SB) model, an improved version of the position-based (IV_PB) model proposed by Ying et al. [9]. This paper also introduces a procedure that combines another mathematical model with a heuristic solution obtained through GRASP to determine an upper bound value for the decision variables in IV_PB model, aiming to eliminate unnecessary extra positions in IV_PB model (determines an upper bound value for the decision variables incorporating position assignments). Subsequently, a new MILP model based on IV_PB model is developed by restricting the decision variables using the obtained upper bound value, referred to as RIV_PB model. The traditional Benders decomposition algorithm is then applied to RIV_PB model. To accelerate the convergence of the Benders algorithm, a hybrid Benders algorithm is developed by combining RIV_PB model and GRASP within the Benders decomposition framework. Experimental results show that the new exact methods introduced in this paper outperform previous approaches in all aspects. Notably, the hybrid Benders algorithm proves to be more

effective than other exact methods. For comparison, IRG is reimplemented in this paper, and the computational results demonstrate that GRASP performs competitively with IRG. In addition to its effectiveness, GRASP is structurally simple and involves very few parameters, making it an ideal choice for such problems. As future work, the hybrid Benders decomposition algorithm and GRASP can be applied to other types of flow-shop scheduling problems.

CRediT authorship contribution statement

Alper Hamzadayı: Software, Methodology, Conceptualization. **Münevver Günay Van:** Writing – review & editing, Writing – original draft, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.swevo.2025.102028.

Data availability

No data was used for the research described in the article.

References

[1] F.T.S. Chan, S.H. Chung, L.Y. Chan, G. Finke, M.K. Tiwari, Solving distributed FMS scheduling problems subject to maintenance: genetic algorithms approach, Robot. Comput.-Integr. Manuf. 22 (2006) 493–504, https://doi.org/10.1016/j.rcim.2005.11.005.

[2] H.Z. Jia, Y.H.J. Fuh, A.Y.C. Nee, Y.F. Zhang, Integration of genetic algorithm and Gantt chart for job shop scheduling in distributed manufacturing systems, Comput. Ind. Eng. 53 (2007) 313–320, https://doi.org/10.1016/j.cie.2007.06.024.

[3] J. Deng, L. Wang, A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem, Swarm Evol. Comput. 32 (2017) 121–131, https://doi.org/10.1016/j.swevo.2016.06.002.

[4] Y. Fu, Y. Hou, Z. Chen, X. Pu, K. Gao, A. Sadollah, Modelling and scheduling integration of distributed production and distribution problems via black widow optimization, Swarm Evol. Comput. 68 (2022) 101015, https://doi.org/10.1016/j.swevo.2021.101015.

[5] B.L. Maccarthy, J. Liu, Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling, Int. J. Prod. Res. 31 (1993) 59–79, https://doi.org/10.1080/00207549308956713.

[6] J.N.D. Gupta, E.F. Stafford, Flowshop scheduling research after five decades, Eur. J. Oper. Res. 169 (2006) 699–711, https://doi.org/10.1016/j.ejor.2005.02.001.

[7] L.D. Giovanni, F. Pezzella, An improved genetic algorithm for the distributed and flexible job-shop scheduling problem, Eur. J. Oper. Res. 200 (2010) 395–408, https://doi.org/10.1016/j.ejor.2009.01.008.

[8] H.-Y. Sang, Q.-K. Pan, J.-Q. Li, P. Wang, Y.-Y. Han, K.-Z. Gao, P. Duan, Effective invasive weed optimization algorithms for distributed assembly permutation flowshop problem with total flowtime criterion, Swarm Evol. Comput. 44 (2019) 64–73, https://doi.org/10.1016/j.swevo.2018.12.001.

[9] K.-C. Ying, S.-W. Lin, C.-Y. Cheng, C.-D. He, Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems, Comput. Ind. Eng. 110 (2017) 413–423, https://doi.org/10.1016/j.cie.2017.06.025.

[10] B. Naderi, R. Ruiz, The distributed permutation flowshop scheduling problem, Comput. Oper. Res. 37 (4) (2010) 754–768, https://doi.org/10.1016/j.cor.2009.06.019.

[11] J. Gao, R. Chen, An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems, Sci. Res. Essays. 6 (14) (2011) 3094–3100, https://doi.org/10.5897/SRE10.1014.

[12] J. Gao, R. Chen, A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem, Int. J. Comput. Intell. Syst. 4 (4) (2011) 497–508, https://doi.org/10.1080/18756891.2011.9727808.

[13] J. Gao, R. Chen, W. Deng, Y. Liu, Solving multi-factory flowshop problems with a novel variable neighbourhood descent algorithm, J. Comput. Inf. Syst. 8 (5) (2012) 2025–2032.

[14] J. Gao, R. Chen, W. Deng, An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (3) (2013) 641–651, https://doi.org/10.1080/00207543.2011.644819.

[15] S.-W. Lin, K.-C. Ying, C.-Y. Huang, Minimising makespan in distributed permutation flowshops using a modified iterated Greedy algorithm, Int. J. Prod. Res. 51 (16) (2013) 5029–5038, https://doi.org/10.1080/00207543.2013.790571.

[16] S.Y. Wang, L. Wang, M. Liu, Y. Xu, An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem, Int. J. Prod. Econ. 145 (1) (2013) 387–396, https://doi.org/10.1016/J.IJPE.2013.05.004.

[17] B. Naderi, R. Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem, Eur. J. Oper. Res. 239 (2) (2014) 323–334, https://doi.org/10.1016/j.ejor.2014.05.024.

[18] Y. Xu, L. Wang, S. Wang, M. Liu, An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem, Eng. Optim. 46 (9) (2014) 1269–1283, https://doi.org/10.1080/0305215X.2013.827673.

[19] V. Fernandez-Viagas, J.M. Framinan, A bounded-search iterated Greedy algorithm for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 53 (4) (2015) 1111–1123, https://doi.org/10.1080/00207543.2014.948578.

[20] H. Bargaoui, O. Belkahla Driss, K. Ghédira, A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion, Comput. Ind. Eng. 111 (2017) 239–250, https://doi.org/10.1016/j.cie.2017.07.020.

[21] R. Ruiz, Q.-K. Pan, B. Naderi, Iterated Greedy methods for the distributed permutation flowshop scheduling problem, OMEGA: Int. J. Manag. Sci. 83 (2019) 213–222, https://doi.org/10.1016/j.omega.2018.03.004.

[22] J.-Q. Li, S.-C. Bai, P.-Y. Duan, H.-Y. Sang, Y.-Y. Han, Z.-X. Zheng, An improved artificial bee colony algorithm for addressing distributed flow shop with distance coefficient in a prefabricated system, Int. J. Prod. Res. 57 (22) (2019) 6922–6942, https://doi.org/10.1080/00207543.2019.1571687.

[23] J.F. Ren, C.M. Ye, Y. Li, A new solution to distributed permutation flow shop scheduling problem based on NASH Q-learning, Adv. Prod. Eng. Manag. 16 (3) (2021) 269–284, https://doi.org/10.14743/apem2021.3.399.

[24] J.-P. Huang, Q.-K. Pan, L. Gao, An effective iterated Greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times, Swarm Evol. Comput. 59 (2020) 100742, https://doi.org/10.1016/j.swevo.2020.100742.

[25] A. Hamzadayı, An effective benders decomposition algorithm for solving the distributed permutation flowshop scheduling problem, Comput. Oper. Res. 123 (2020) 105006, https://doi.org/10.1016/j.cor.2020.105006.

[26] Q.K. Pan, R. Ruiz, An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem, Omega 44 (1) (2014) 41–50, https://doi.org/10.1016/j.omega.2013.10.002.

[27] N.E.H. Saadani, A. Guinet, M. Moalla, Three stage no-idle flow-shops, Comput. Ind. Eng. 44 (3) (2003) 425–434, https://doi.org/10.1016/S0360-8352(02)00217-6.

[28] M.F. Tasgetiren, Q.K. Pan, P.N. Suganthan, O. Buyukdagli, A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem, Comput. Oper. Res. 40 (7) (2013) 1729–1743, https://doi.org/10.1016/j.cor.2013.01.005.

[29] I. Adiri, D. Pohoryles, Flowshop/no-idle or no-wait scheduling to minimize the sum of completion times, Nav. Res. Logist. Q. 29 (3) (1982) 495–504, https://doi.org/10.1002/nav.3800290311.

[30] P.J. Kalczynski, J. Kamburowski, A heuristic for minimizing the makespan in no-idle permutation flow shops, Comput. Ind. Eng. 49 (1) (2005) 146–154, https://doi.org/10.1016/j.cie.2005.05.002.

[31] Q.K. Pan, L. Wang, A novel differential evolution algorithm for no-idle permutation flow-shop scheduling problems, Eur. J. Ind. Eng. 2 (3) (2008) 279–297.

[32] G. Deng, X. Gu, A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion, Comput. Oper. Res. 39 (9) (2012) 2152–2160, https://doi.org/10.1016/j.cor.2011.10.024.

[33] R. Ruiz, T. Stutzle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, Eur. J. Oper. Res. 177 (3) (2007) 2033–2049, https://doi.org/10.1016/j.ejor.2005.12.009.

[34] S.F. Rad, R. Ruiz, N. Boroojerdian, New high performing heuristics for minimizing makespan in permutation flowshops, Omega 37 (2) (2009) 331–345, https://doi.org/10.1016/j.omega.2007.02.002.

[35] K.C. Ying, Solving non-permutation flowshop scheduling problems by an effective iterated greedy heuristic, Int. J. Adv. Manuf. Technol. 38 (3–4) (2008) 348–354, https://doi.org/10.1007/s00170-007-1104-y.

[36] K.C. Ying, Minimising makespan for multistage hybrid flowshop scheduling problems with multiprocessor tasks by a hybrid immune algorithm, Eur. J. Ind. Eng. 6 (2) (2012) 199–215, https://doi.org/10.1504/EJIE.2012.045605.

[37] K.C. Ying, Scheduling identical wafer sorting parallel machines with sequence-dependent setup times using an iterated greedy heuristic, Int. J. Prod. Res. 50 (10) (2012) 2710–2719, https://doi.org/10.1080/00207543.2011.588617.

[38] K.C. Ying, S.W. Lin, S.Y. Wan, Bi-objective reentrant hybrid flowshop scheduling: an iterated Pareto greedy algorithm, Int. J. Prod. Res. 52 (19) (2014) 5735–5747, https://doi.org/10.1080/00207543.2014.910627.

[39] S. Hatami, R. Ruiz, C. Andrés-Romano, Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times, Int. J. Prod. Econ. 169 (1) (2015) 76–88, https://doi.org/10.1016/j.ijpe.2015.07.027.

[40] E. Vallada, R. Ruiz, J.M. Framinan, New hard benchmark for flowshop scheduling problems minimising makespan, Eur. J. Oper. Res. 240 (3) (2015) 666–677, https://doi.org/10.1016/j.ejor.2014.07.033.

[41] C.-Y. Cheng, K.-C. Ying, H.-H. Chen, H.-S. Lu, Minimising makespan in distributed mixed no-idle flowshops, Int. J. Prod. Res. 57 (1) (2019) 48–60, https://doi.org/10.1080/00207543.2018.1457812.

[42] J.-F. Chen, L. Wang, Z.-P. Peng, A collaborative optimization algorithm for energy-efficient multi-objective distributed no-idle flow-shop scheduling, Swarm Evol. Comput. 50 (2019) 100557, https://doi.org/10.1016/j.swevo.2019.100557.

[43] Y.-Z. Li, Q.-K. Pan, J.-Q. Li, L. Gao, M.F. Tasgetiren, An adaptive iterated Greedy algorithm for distributed mixed no-idle permutation flowshop scheduling problems, Swarm Evol. Comput. 63 (2021) 100874, https://doi.org/10.1016/j.swevo.2021.100874.
[44] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (2) (1979) 287–362, https://doi.org/10.1016/S0167-5060(08)70356-X.
[45] P. Perez-Gonzalez, J.M. Framinan, A review and classification on distributed permutation flowshop scheduling problems, Eur. J. Oper. Res. 312 (2024) 1–21, https://doi.org/10.1016/j.ejor.2023.02.001.
[46] T.A. Feo, M.G.C. Resende, S.H. Smith, A greedy randomized adaptive search procedure for maximum independent set, Oper. Res. 42 (5) (1994) 860–878, https://doi.org/10.1287/opre.42.5.860.
[47] T.A. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures, J. Glob. Optim. 6 (2) (1995) 109–133, https://doi.org/10.1007/BF01096763.
[48] M. Avci, M.G. Avci, A. Hamzadayı, A branch-and-cut approach for the distributed no-wait flowshop scheduling problem, Comput. Oper. Res. 148 (2022) 106009, https://doi.org/10.1016/j.cor.2022.106009.
[49] L.C. Coelho, G. Laporte, Improved solutions for inventory-routing problems through valid inequalities and input ordering, Int. J. Prod. Econ. 155 (2014) 391–397, https://doi.org/10.1016/j.ijpe.2013.11.019.
[50] H.D. Sherali, J.C. Smith, Improving discrete model representations via symmetry considerations, Manag. Sci. 47 (10) (2001) 1396–1407, https://doi.org/10.1287/mnsc.47.10.1396.10265.
[51] Z. Degraeve, W. Gochet, R. Jans, Alternative formulations for a layout problem in the fashion industry, Eur. J. Oper. Res. 143 (1) (2002) 80–93, https://doi.org/10.1016/S0377-2217(01)00330-7.
[52] R. Jans, Solving lot-sizing problems on parallel identical machines using symmetry-breaking constraints, INFORMS J. Comput. 21 (1) (2009) 123–136, https://doi.org/10.1287/ijoc.1080.0283.
[53] Y. Adulyasak, J.-F. Cordeau, R. Jans, Formulations and branch-and-cut algorithms for multivehicle production and inventory routing problems, INFORMS J. Comput. 26 (1) (2014) 103–120, https://doi.org/10.1287/ijoc.2013.0550.
[54] J.F. Benders, Partitioning procedures for solving mixed-variables programming problems, Numer. Math. 4 (1) (1962) 238–252, https://doi.org/10.1007/BF01386316.
[55] R. Rahmaniani, T.G. Crainic, M. Gendreau, W. Rei, The Benders decomposition algorithm: a literature review, Eur. J. Oper. Res. 259 (3) (2017) 801–817, https://doi.org/10.1016/j.ejor.2016.12.005.
[56] H. Mokhtar, M. Krishnamoorthy, A.T. Ernst, The 2-allocation p-hub median problem and a modified Benders decomposition method for solving hub location problems, Comput. Oper. Res. 104 (2019) 375–393, https://doi.org/10.1016/j.cor.2018.09.006.
[57] Z.C. Taşkın, M. Cevik, Combinatorial Benders cuts for decomposing IMRT fluence maps using rectangular apertures, Comput. Oper. Res. 40 (9) (2013) 2178–2186, https://doi.org/10.1016/j.cor.2011.07.005.
[58] A. Moreno, P. Munari, D. Alem, Decomposition-based algorithms for the crew scheduling and routing problem in road restoration, Comput. Oper. Res. 119 (2020) 104935, https://doi.org/10.1016/j.cor.2020.104935.
[59] T. Bektaş, A. Hamzadayı, R. Ruiz, Benders decomposition for the mixed no-idle permutation flowshop scheduling problem, J. Sched. 23 (2020) 513–523, https://doi.org/10.1007/s10951-020-00637-8.
[60] M.G. Resende, C.C. Ribeiro, Greedy randomized adaptive search procedures: advances, hybridizations, and applications. Handbook of Metaheuristics, Springer, 2010, pp. 283–319, https://doi.org/10.1007/978-1-4419-1665-5_10.
[61] F. Palmieri, U. Fiore, S. Ricciardi, A. Castiglione, GRASP-based resource re-optimization for effective big data access in federated clouds, Fut. Gener. Comput. Syst. 54 (2016) 168–179, https://doi.org/10.1016/j.future.2015.01.017.
[62] A. Exposito-Marquez, C. Exposito-Izquierdo, J. Brito-Santana, J.A. Moreno-Pérez, Greedy randomized adaptive search procedure to design waste collection routes in La Palma, Comput. Ind. Eng. 137 (2019) 106047, https://doi.org/10.1016/j.cie.2019.106047.
[63] A. Baykasoğlu, F.S. Madenoglu, A. Hamzadayi, Greedy randomized adaptive search for dynamic flexible job-shop scheduling, J. Manuf. Syst. 56 (2020) 425–451, https://doi.org/10.1016/j.jmsy.2020.06.005.
[64] L.S. Almeida, F. Goerlandt, R. Pelot, K. Sörensen, A greedy randomized adaptive search procedure (GRASP) for the multi-vehicle prize collecting arc routing for connectivity problem, Comput. Oper. Res. 143 (2022) 105804, https://doi.org/10.1016/j.cor.2022.105804.
[65] J.H. Holland, Adaptation in natural and artificial systems, University of Michigan Press, Ann Arbor, Michigan, 1975.
[66] I. Ribas, R. Companys, X. Tort-Martorell, Comparing three-step heuristics for the permutation flow shop problem, Comput. Oper. Res. 37 (12) (2010) 2062–2070, https://doi.org/10.1016/j.cor.2010.02.006.
[67] E. Taillard, Benchmarks for basic scheduling problems, Eur. J. Oper. Res. 64 (2) (1993) 278–285, https://doi.org/10.1016/0377-2217(93)90182-M.
[68] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evol. Comput. 1 (1) (2011) 3–18, https://doi.org/10.1016/j.swevo.2011.02.002.