algorithms

MDPI

Article

A Variable Block Insertion Heuristic for Solving Permutation Flow Shop Scheduling Problem with Makespan Criterion

Damla Kizilay [1], Mehmet Fatih Tasgetiren [2,3,*], Quan-Ke Pan [4] and Liang Gao [3]

1 Department of Industrial Engineering, Yasar University, Izmir 35100, Turkey; damla.kizilay@yasar.edu.tr
2 Department of Industrial and System Engineering, Istinye University, Istanbul 34010, Turkey
3 Department of Industrial and Manufacturing System Engineering, Huazhong University of Science and Technology, Wuhan 430074, China; gaoliang@mail.hust.edu.cn
4 School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200444, China; panquanke@shu.edu.cn
* Correspondence: fatih.tasgetiren@istinye.edu.tr; Tel.: +90-530-827-3715

Received: 8 April 2019; Accepted: 6 May 2019; Published: 9 May 2019

Abstract: In this paper, we propose a variable block insertion heuristic (VBIH) algorithm to solve the permutation flow shop scheduling problem (PFSP). The VBIH algorithm removes a block of jobs from the current solution. It applies an insertion local search to the partial solution. Then, it inserts the block into all possible positions in the partial solution sequentially. It chooses the best one amongst those solutions from block insertion moves. Finally, again an insertion local search is applied to the complete solution. If the new solution obtained is better than the current solution, it replaces the current solution with the new one. As long as it improves, it retains the same block size. Otherwise, the block size is incremented by one and a simulated annealing-based acceptance criterion is employed to accept the new solution in order to escape from local minima. This process is repeated until the block size reaches its maximum size. To verify the computational results, mixed integer programming (MIP) and constraint programming (CP) models are developed and solved using very recent small VRF benchmark suite. Optimal solutions are found for 108 out of 240 instances. Extensive computational results on the VRF large benchmark suite show that the proposed algorithm outperforms two variants of the iterated greedy algorithm. 236 out of 240 instances of large VRF benchmark suite are further improved for the first time in this paper. Ultimately, we run Taillard's benchmark suite and compare the algorithms. In addition to the above, three instances of Taillard's benchmark suite are also further improved for the first time in this paper since 1993.

Keywords: heuristic optimization; block insertion heuristic; flow shop scheduling; iterated greedy algorithm; constraint programming; mixed integer programming

1. Introduction

Sustainability in manufacturing industries is mainly measured by their competitiveness in the market place. Competitiveness is referred to timely product delivery with the best quality, minimum manufacturing time and price to customers. Minimum manufacturing time can be obtained by optimal production sequences that can minimize makespan or total flowtime. Note that a manufacturing company can fail to satisfy production plans although the other production entities such as operators, maintenance, inventory, quality control, etc. are in control due to the lack of optimal or near optimal production sequences in the shop floor. For this reason, seeking optimal or near-optimal production sequences and schedules is vital to manufacturing companies in order to minimize the makespan, which also minimizes idle times on the machines and maximize machine utilization.

The permutation flow shop scheduling problem (PFSP) has been widely studied in the literature and has extensively been applied in the inustry. There are many different fields in real-life where PFSP can be used [1]. It is yet an exceptionally active topic of investigation, especially because flow shop environments are at the center of real-life scheduling problems in various fields of high social or economic impact. In addition, the flow shop layout is a regular configuration in many manufacturing companies. The basic PFSP consists of a set of n jobs which are processed by m machines. These jobs follow the same route and their operations on the machines cannot be interrupted. All the jobs must be processed in the same order on the machines and the aim is to find the best permutation $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ of these jobs with respect to the given objective.

In this study, our aim is to maximize the throughput of the system by maximizing the utilization rate of the machines which means minimizing makespan. To compute the makespan, π denotes the given arbitrary solution, where job π_i is the job at the ith position of solution π. $C_{i,k}$ is denoted as the completion time of job π_i on machine k at position i. Following this notation, completion times of jobs at each machine are computed as in the following Equations (1)–(5), where $p_{\pi_i,k}$ is the processing time of job π_i at the kth machine. The makespan of solution π, denoted as $C_{max}(\pi)$, is the completion time of the last job (i.e., n) on the last machine (i.e., m). It is simply denoted as C_{nm} and calculated as follows:

$$C_{1,1} = p_{\pi_1,1} \tag{1}$$

$$C_{i,1} = C_{i-1,1} + p_{\pi_i,1} \forall i = 2, \ldots, n \tag{2}$$

$$C_{1,k} = C_{1,k-1} + p_{\pi_1,k} \forall k = 2, \ldots, m \tag{3}$$

$$C_{i,k} = max\Big\{C_{i-1,k},\ C_{i,k-1}\Big\} + p_{\pi_i,k} \forall i = 2, \ldots, n; \forall k = 2, \ldots, m \tag{4}$$

$$C_{max}(\pi) = C_{nm}. \tag{5}$$

The PFSP with makespan criterion is denoted as *Fm*|*Permu*|*Cmax* according to the notation of [2] and has been proven to be NP-hard for the makespan criterion [3], so it is challenging to solve it with exact methods. Therefore, metaheuristic algorithms were employed to solve the problem and obtain near-optimal solutions. In recent years, various metaheuristic algorithms have been presented to solve various variants of PFSP with different objectives. One of the state-of-the-art algorithms for PFSPs is the iterated greedy algorithm (IG) presented by [4]. We focused on the recent literature that considers the IG algorithm in their solution approaches.

The IG algorithm was employed to PFSP with makespan criterion in [4–9]. In [5], to improve the solution quality, a local search was applied to the partial solution after the destruction step of the IG algorithm, while in [6] sequence depended setup times were employed for the PFSP with makespan criterion. In addition, in [7] the authors studied the PFSP with makespan and proposed a tie-breaking mechanism for the IG algorithm, while in [8] an IG and a discrete differential evolution algorithm were proposed and compared. In this study, we employ new hard VRF instances which are first introduced in [9], and they also applied an IG algorithm. In addition, the same problem was studied in [10] to minimize the makespan over Taillard's benchmark suite.

The IG algorithm was applied to various variants of PFSP such as no-wait flow shops in [11–13]; blocking flow shops in [14–17]; no-idle flow shops in [18–20]; energy-efficient PFSP in [21,22]; multi-objective PFSP in [23,24] where both studies presented a restarted iterated Pareto greedy algorithm. In a no-wait variant of PFSP, distributed no-wait flow shop problem [11], tabu-based reconstruction strategy [12], and sequence depended setup times [13] were employed with IG algorithm. In blocking variant of PFSP, IG algorithms were combined with local search algorithms [14], constructive heuristics [15,16], and also embedded in differential evolution framework [17]. In [25] profile fitting and NEH heuristic algorithms were proposed for the same problem. In a no-idle variant of PFSP, iterated reference greedy algorithm [18], and variable IG algorithm [19] were presented. In addition, IG algorithm was employed for the mixed no-idle PFSP [20].

IG algorithm was also applied to PFSP with different objective functions such as total tardiness criterion [26,27]; total flowtime criterion [28]. In [1], they carried out an exhaustive review and computational evaluation of heuristics and metaheuristics published until 2017 for the PFSP to minimize the makespan. Therefore, for the further analysis of the literature of PFSP, the indicated manuscript [1] should be examined.

In traditional search algorithms, swap and insertion neighborhood structures are generally employed. The swap neighborhood exchanges two jobs in a solution, whereas the insertion one removes a job from a solution and inserts it into another position in the solution. Recently, block move-based local search algorithms are presented for the single machine-scheduling problems in the literature [29–32]. Xu et al. [31] developed a *Block Move* neighborhood structure in which l consecutive jobs (called a block) are inserted into another position in the solution. They represent a block move by a *triplet* $(i,\ k,\ l)$, where i denotes the position of the first job of the block, k the target position of the block to be inserted and l the size of the block. It is obvious that one edge insertion, two edge-insertion and 3-block insertion are the block move neighborhoods with $l = 1$, $l = 2$, and $l = 3$. Similarly, Gonzales and Vela [32] developed a variable neighborhood descent algorithm with three distinct block move neighborhoods and employed in a memetic algorithm. Then, a memetic algorithm with block insertion heuristic is presented in [29]. Moreover, in [33], a variable block insertion heuristic (VBIH) algorithm was employed to solve the blocking PFSP with makespan criterion.

In IG algorithms, some solutions components are removed from the current solution and reinserted into the partial solutions. In other words, a number, dS, of jobs are removed randomly, which is known as the destruction phase. Then, in the construction phase, these dS jobs are reinserted into the partial solution in the same order they are removed. For each of dS jobs, it makes a number $n - dS + 1$ of insertions. However, the VBIH algorithm removes a block of jobs π_b with size b from the current solution and it makes a number $n - b + 1$ of block insertions only. That is the difference between IG and VBIH algorithms.

The main contributions of the paper can be outlined as follows. VBIH is employed to solve PFSP with makespan criterion using the new hard VRF benchmark sets [9]. Detailed computational results show that VBIH algorithm outperforms two variants of the iterated greedy algorithm. 236 out of 240 instances of large VRF benchmark suite are further improved for the first time in this paper, while the results of the remaining four instances are found as the same with the current results. In addition, the formulation of two mathematical models is given to solve the small benchmark set in order to verify the results of our proposed VBIH algorithm. One hundred and eight out of 240 small instances are proven to be optimal. Therefore, this paper proposes new lower bounds with the use of an efficient algorithm, which differentiates the study from the current literature. We also show that the speed up method of Taillard is substantially effective when solving the PFSP with makespan criterion.

The remaining part of the paper is organized as follows: Section 2 introduces the formulation of PFSP including mixed integer programming (MIP) model and constraint programming (CP) model whereas Section 3 presents all the heuristic algorithms. Section 4 explains the computational results of the MIP and CP models on small VRF instances to show the solution quality of the heuristic algorithms and the limitations of the models. Section 5 reports the experimental results of the heuristic algorithms and the improvements to the large VRF instances. Finally, Section 6 summarizes the concluding remarks.

2. Mathematical Model Formulation

This paper proposes MIP and CP models to solve small VRF instances for PFSP with the makespan criterion in order to verify the solution quality of proposed heuristic algorithms. The input parameters used in the models are presented in follows:

Parameters:

n: Total number of jobs, $i = 1, \ldots, n$
m: Total number of machines, $k = 1, \ldots, m$
$p_{i,k}$: Processing time of job i on machine k
M : A sufficient large constant integer.

2.1. The MIP Model

The MIP decision variables, objective function and the constraints are given in the following equations. The MIP formulation of PFSP, which were proposed by Manne [34], is used.

Decision Variables:

C_{max}: Makespan
$C_{i,k}$: Completion time of job i on machine k
$D_{i,j}$: Binary variable: 1 if job i is scheduled before job j; 0, otherwise; $i < j$

MIP Model: Objective and Constraints:

$$\begin{gathered} Min\ C_{max} \\ st: \end{gathered} \tag{6}$$

$$C_{max} \geq C_{i,m} \forall i = 1, \ldots, n \tag{7}$$

$$C_{i,1} \geq p_{i,1} \forall i = 1, \ldots, n \tag{8}$$

$$C_{i,k} - C_{i,k-1} \geq p_{i,k} \forall i = 1, \ldots, n,\ \forall k = 2, \ldots, m \tag{9}$$

$$C_{i,k} - C_{j,k} + MD_{i,j} \geq p_{i,k} \forall i = 1 \leq i < j \leq n,\ \forall k = 1, \ldots, m \tag{10}$$

$$C_{i,k} - C_{j,k} + MD_{i,j} \leq M - p_{j,k} \forall i = 1 \leq i < j \leq n,\ \forall k = 1, \ldots, m \tag{11}$$

$$D_{i,j} \in (0,1). \tag{12}$$

The objective function (6) minimizes the makespan while Constraint (7) calculates the maximum completion time of all jobs on the last machine. In PFSP, all jobs follow the same route through the machines so that their final processes will be done on the last machine. Constraint (8) computes the completion time of each job on machine 1 ensuring that they cannot occur earlier than the duration of their processing time on machine 1 which is the starting machine for all jobs. Constraint (9) ensures that the completion time of each job on each machine cannot be processed before their completion time on the previous machine. Constraints (10) and (11) specify the relationship between the processing of two consecutive jobs on the same machine. Constraint (11) starts that if job i precedes job j in the permutation, then job i should be completed before job j on each machine. Otherwise, job j should precede job i on each machine which is shown by Constraint (10).

2.2. The CP Model

CP decision variables, objective function and the constraints are presented in the following equations using the OPL API of CP Optimizer. To express the processing times of the jobs on the machines, the model uses interval variables denoted as *JobInt*. In addition, sequence variables for the machines are defined in the model as *Machine* which collects all these interval variables.

Decision Variables:

$JobInt_{i,k}$: Interval variable for job i on machine k with duration $p_{i,k}$
$Machine_k$: Sequence variable for machine k over $\left\{JobInt_{i,k} \middle| 1 \leq i \leq n\right\}$.

CP Model: Objective and Constraints:

$$Min\left(\max_{i\in J}(endOf(JobInt_{i,m}))\right) \tag{13}$$

$$endBeforeStart(JobInt_{i,k}, JobInt_{i,k+1})\forall i = 1,\ldots,n,\ \forall k = 1,\ldots,m-1 \tag{14}$$

$$noOverlap(Machine_k)\forall k = 1,\ldots,m \tag{15}$$

$$sameSequence(Machine_1, Machine_k)\forall k = 2,\ldots,m. \tag{16}$$

The CP model minimizes the makespan by computing the maximum end date of each job on the last machine (13). Constraint (14) impose the precedence constraints between the consecutive operations of each job on the sequence of machines. Machines are the disjunctive resources and can process only one job at a time, which is expressed by the *noOverlap* Constraint (15) over the sequence variables associated with machines. The relationship between sequence variables and the interval variables are provided while defining the decision variables. The last constraint *sameSequence* (16) guarantees that all the jobs are processed in the same order on each machine. Therefore, the permutation of the jobs will be the same for each machine.

3. Meta-Heuristic Algorithms

3.1. Taillard's Speed Up Method for PFSP with Makespan Criterion

Insertion neighborhood structure is very effective for makespan minimization. The size of the insertion neighborhood is $(n-1)^2$. Since each objective function evaluation takes $O(nm)$ time, its computational complexity is $O(n^3m)$. In [35], a speed-up method is proposed where it reduces the computational complexity from $O(n^3m)$ to $O(n^2m)$ for the PFSP with makespan criterion. In order to execute the insertion procedure in time $O(nm)$, this speed-up method can be explained as follows: Suppose that job π_i will be inserted in a position l. Then the speed up method can be described below:

1. Compute the head, $e_{i,k}$, which is the earliest completion time of each job on each machine. The starting time of the first job on the first machine is 0. $e_{0,k} = e_{i,0} = 0$ $\forall i = 1,\ldots,l-1; \forall k = 1,\ldots,m$ $e_{i,k} = max\left\{e_{i,k-1}, e_{i-1,k}\right\} + p_{\pi_i,k}$ $\forall i = 1,\ldots,l-1; \forall k = 1,\ldots,m.$
2. Compute the tail, $q_{i,k}$, which is the duration between the starting time of each job on each machine and the end of all the operations on each machine. $q_{i,m+1} = 0$ $\forall i = n,\ldots,l-1;\ \forall k = m,\ldots,1$ $q_{l,k} = 0$ $\forall i = n,\ldots,l-1;\ \forall k = m,\ldots,1$ $q_{i,k} = max\left\{q_{i,k+1}, q_{i+1,k}\right\} + p_{\pi_i,k}$ $\forall i = n,\ldots,l-1;\ \forall k = m,\ldots,1.$
3. Compute the earliest relative completion time $f_{i,k}$ on the lth machine of job π_j inserted at the lth position. Completion time of an inserted job on the first machine is zero. $f_{i,0} = 0$ $\forall i = 1,\ldots,l$ $f_{i,k} = max\left\{f_{i,k-1}, e_{i-1,k}\right\} + p_{\pi_i,k}$ $\forall i = 1,\ldots,l; \forall k = 1,\ldots,m.$
4. The value of the makespan $C_{max,l}$ when inserting job j at the lth position is: $C_{max,l} = max_k(f_{ik} + q_{ik})$ $\forall i = 1,\ldots,l; \forall k = 1,\ldots,m.$

In order to illustrate the speed up the procedure, we give the 7-job 2-machine example. Note that Johnson's algorithm [36] solves this problem to optimality. Hence, in Table 1, we provide the problem instance with the processing times as well as the optimal solution.

Table 1. Problem instance with processing times and optimal solution.

Instance			Optimal Solution with C_{max}=36			
Jobs	**M1**	**M2**	**Jobs**	**Position**	**M1**	**M2**
1	1	8	1	1	1	8
2	2	9	2	2	2	9
3	7	5	7	3	4	5
4	5	3	3	4	7	5
5	5	4	5	5	5	4
6	7	1	4	6	5	3
7	4	5	6	7	7	1

According to the Johnson's algorithm [36], the optimal solution is $\{1,2,7,3,5,4,6\}$ with $C_{max} = 36$. Now, suppose that we remove job 7 and obtain the partial solution, $\{1,2,3,5,4,6\}$. Suppose that we insert job 7 into position $l = 3$ of the partial solution to obtain the optimal solution. We follow the speed up method now:

1. Compute heads:

$$e_{0,k} = e_{i,0} = 0 \qquad \forall i = 1,\ldots,l-1; \forall k = 1,\ldots,m$$

$$e_{i,k} = max\left\{e_{i,k-1}, e_{i-1,k}\right\} + p_{\pi_i,k} \qquad \forall i = 1,\ldots,l-1; \forall k = 1,\ldots,m$$

$$e_{1,1} = max\left\{e_{1,0}, e_{0,1}\right\} + p_{\pi_1,1} = max\left\{e_{1,0}, e_{0,1}\right\} + p_{1,1} = max\{0,0\} + 1 = 1$$

$$e_{1,2} = max\left\{e_{1,1}, e_{0,2}\right\} + p_{\pi_1,2} = max\left\{e_{1,1}, e_{0,2}\right\} + p_{1,2} = max\{1,0\} + 8 = 9$$

$$e_{2,1} = max\left\{e_{2,0}, e_{1,1}\right\} + p_{\pi_2,1} = max\left\{e_{2,0}, e_{1,1}\right\} + p_{2,1} = max\{0,1\} + 2 = 3$$

$$e_{2,2} = max\left\{e_{2,1}, e_{1,2}\right\} + p_{\pi_2,2} = max\left\{e_{2,1}, e_{1,2}\right\} + p_{2,2} = max\{3,9\} + 9 = 18.$$

2. Compute tails:

$$q_{i,m+1} = 0 \qquad \forall i = n,\ldots,l-1;\ \forall k = m,\ldots,1$$

$$q_{l,k} = 0 \qquad \forall i = n,\ldots,l-1;\ \forall k = m,\ldots,1$$

$$q_{i,k} = max\left\{q_{i,k+1}, q_{i+1,k}\right\} + p_{\pi_i,k} \qquad \forall i = n,\ldots,l-1;\ \forall k = m,\ldots,1$$

$$q_{6,2} = max\{q_{6,3}, q_{7,2}\} + p_{\pi_6,2} = max\{q_{6,3}, q_{7,2}\} + p_{6,2} = max\{0,0\} + 1 = 1$$

$$q_{6,1} = max\left\{q_{6,2}, q_{7,1}\right\} + p_{\pi_6,1} = max\left\{q_{6,2}, q_{7,1}\right\} + p_{6,1} = max\{1,0\} + 7 = 8$$

$$q_{5,2} = max\{q_{5,3}, q_{6,2}\} + p_{\pi_5,2} = max\{q_{5,3}, q_{6,2}\} + p_{4,2} = max\{0,1\} + 3 = 4$$

$$q_{5,1} = max\left\{q_{5,2}, q_{6,1}\right\} + p_{\pi_5,1} = max\left\{q_{5,2}, q_{6,1}\right\} + p_{4,1} = max\{4,8\} + 5 = 13$$

$$q_{4,2} = max\left\{q_{4,3}, q_{5,2}\right\} + p_{\pi_4,2} = max\left\{q_{4,3}, q_{5,2}\right\} + p_{5,2} = max\{0,4\} + 4 = 8$$

$$q_{4,1} = max\left\{q_{4,2}, q_{5,1}\right\} + p_{\pi_4,1} = max\left\{q_{4,2}, q_{5,1}\right\} + p_{5,1} = max\{8,13\} + 5 = 18$$

$$q_{3,2} = max\left\{q_{3,3}, q_{4,2}\right\} + p_{\pi_3,2} = max\left\{q_{3,3}, q_{4,2}\right\} + p_{3,2} = max\{0,8\} + 5 = 13$$

$$q_{3,1} = max\left\{q_{3,2}, q_{4,1}\right\} + p_{\pi_3,1} = max\left\{q_{3,2}, q_{4,1}\right\} + p_{3,1} = max\{13,18\} + 7 = 25.$$

Speed-up calculation of the partial solution is given in Figure 1.

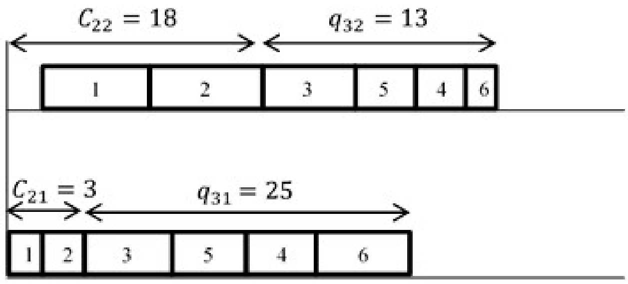

Figure 1. Speed-up calculation of a partial solution.

5. Compute the earliest relative completion time $f_{i,k}$

$$f_{i,0} = 0 \qquad \forall i = 1, \ldots, l$$

$$f_{i,k} = max\{f_{i,k-1}, e_{i-1,k}\} + p_{\pi_i,k} \qquad \forall i = 1, \ldots, l; \forall k = 1, \ldots, m$$

$$f_{1,1} = max\{f_{1,0}, e_{0,1}\} + p_{\pi_1,1} = max\{f_{1,0}, e_{0,1}\} + p_{1,1} = max\{0, 0\} + 1 = 1$$

$$f_{1,2} = max\{f_{1,1}, e_{0,2}\} + p_{\pi_1,2} = max\{f_{1,1}, e_{0,2}\} + p_{1,2} = max\{1, 0\} + 8 = 9$$

$$f_{2,1} = max\{f_{2,0}, e_{1,1}\} + p_{\pi_2,1} = max\{f_{2,0}, e_{1,1}\} + p_{2,1} = max\{0, 1\} + 2 = 3$$

$$f_{2,2} = max\{f_{2,1}, e_{1,2}\} + p_{\pi_2,2} = max\{f_{2,1}, e_{1,2}\} + p_{2,2} = max\{3, 9\} + 9 = 18$$

$$f_{3,1} = max\{f_{3,0}, e_{2,1}\} + p_{\pi_3,1} = max\{f_{3,0}, e_{2,1}\} + p_{7,1} = max\{0, 3\} + 4 = 7$$

$$f_{3,2} = max\{f_{3,1}, e_{2,2}\} + p_{\pi_3,2} = max\{f_{3,1}, e_{2,2}\} + p_{7,2} = max\{7, 18\} + 5 = 23.$$

Speed-up calculation of the complete solution is given in Figure 2.

6. The value of the makespan $C_{max,l}$ when inserting job π_i at the lth position is:

$$C_{max,l} = max_k(f_{ik} + q_{ik}) \qquad i = l; \forall k = 1, \ldots, m$$

$$C_{max,3} = max_k(f_{ik} + q_{ik})$$

$$C_{max,3} = max\{(f_{31} + q_{31}), (f_{32} + q_{32})\}$$

$$C_{max,3} = max\{(7 + 25), (23 + 13)\}$$

$$C_{max} = max\{32, 36\} = 36.$$

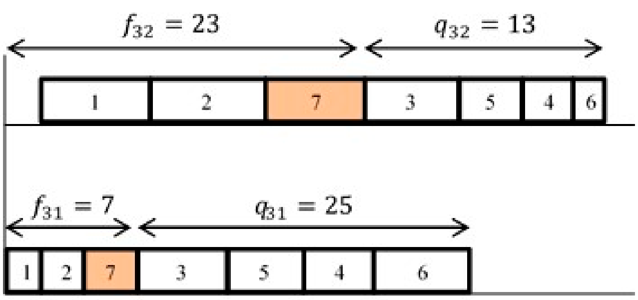

Figure 2. Speed-up calculation of a complete solution.

It is clear that the above speed-up method reduces the complexity of the whole insertion neighborhood from $O(n^3m)$ to $O(n^2m)$. This speed-up method is the key to success for any algorithm for PFSP with makespan criterion. For this reason, we have chosen the Car8 instance from the literature in order to illustrate the speed-up method above in detail. From the literature, we know that best or optimal solution is $\{7, 3, 8, 5, 2, 1, 6, 4\}$ with $C_{max} = 8366$. In Appendix A, we remove job 2 from the optimal solution and re-insert it into the 5th position again. A detailed implementation of Taillard's speed up method is given in Appendix A in order to ease the understanding of it.

3.2. IG Algorithms

IG algorithms mainly have four components; namely, initial solution, destruction-construction (DC) procedure, local search, and acceptance criterion. The traditional IG_{RS} is proposed by [4]. In this algorithm, the initial solution is constructed by the NEH heuristic in [37]. In the destruction step, dS jobs are randomly removed from the solution π without repetition and stored in π_D. The remaining jobs are also stored in π_P that represents the partial solution. In the construction step, each job in π_D is inserted into the partial solution π_P, in the order in which they were removed, until a complete solution of n jobs is constructed. Having carried out the destruction and construction procedure, a local search is employed to further enhance solution quality. After a local search, if the solution is better than or equal to the incumbent solution, it is accepted. Otherwise, it is accepted with a simple simulated annealing-type acceptance criterion, which is suggested by [38]:

$$T = \frac{\sum_{j=1}^{n} \sum_{k=1}^{m} p_{kj}}{10nm} \times \tau P \tag{17}$$

where τP is a parameter to be adjusted. The pseudo-code of the traditional IG_{RS} is given in Algorithm 1, where r is a uniform random number between 0 and 1.

Algorithm 1: Traditional IG_{RS} algorithm

$\pi = NEH$
$\pi^{best} = \pi$
$while\ (NotTermination)\ do$
$\pi_D = Destruction(\pi)$
$\pi^1 = \text{Construction}(\pi_D, \pi_P)$
$\pi^1 = LocalSearch(\pi^1)$ $//Algorithm\ 4$
$if\ (f(\pi^1) \le f(\pi))then$
$\pi = \pi^1$
$if\ (f(\pi^1) < f(\pi^{best}))then$
$\pi^{best} = \pi^1$
$endif$
$elseif\ (r < exp\{-(f(\pi^1) - f(\pi))/T\})\ then$
$\pi = \pi^1$
$endif$
$endwhile$
$return\ \pi^{best}\ and\ f(\pi^{best})$

The IG_{RS} algorithm for the PFSP under makespan minimization employs an initial solution generated by the NEH heuristic. In addition, the NEH heuristic was extended to the FRB5 heuristic with a local search on the partial solutions [39]. Both heuristics are simple and very effective for minimizing the makespan, and its pseudo-code is given in Algorithm 2. In the first phase, the sum of the processing times on all machines are calculated for each job. Then, jobs are sorted in decreasing order to obtain δ. In the second phase, the first job in δ is selected to establish a partial solution π_1. The remaining jobs in δ are inserted in the partial solution one by one. After each iteration, optionally, a local search is applied to the partial solution. Local search is implemented as long as the partial solution is improved. After having inserted all jobs, a complete solution is obtained. Note that the NEH heuristic is denoted as FRB5 heuristic with an optional local search to partial solutions.

Algorithm 2: NEH and FRB5 constructive heuristics

$\delta = DecreasingOrder(\sum_{k=1}^{m} p_{ik})$
$\pi_1 = \delta_1$
$for\ i = 2\ to\ n\ do$
$\pi_i = InsertJobInBestPosition(\pi_i, \delta_i)$
$\pi_i = ApplyLocalSearch(\pi_i,\ f(\pi_i))$ $//Algorithm\ 3\ for\ FRB5\ heuristic$
$end\ for$
$return\ \pi\ with\ n\ jobs\ and\ f(\pi)$

The IG_{RS} algorithm employs insertion neighborhood structure as a local search after destruction and construction procedure. Insertion neighborhood is very effective with the speed-up method explained in Section 3.1 for makespan minimization. Insertion neighborhood can be deterministic or stochastic depending on the decision of choosing a job from solution to be removed. The deterministic variant is given in Algorithm 3. This procedure removes π_i from the solution π and inserts it into all possible positions of the incumbent solution π. When the best-improving insertion position is found, job π_i is inserted into that position. These steps are repeated for all jobs. If an improvement is observed, the local search is re-run until no better solution is obtained.

Algorithm 3: First improvement insertion neighborhood(π)

$for\ i = 1\ to\ n\ do$
$\pi^* = InsertJobInBestPosition(\pi,\ \pi_i)$
$if\ (f(\pi^*) < f(\pi))\ then\ do$
  $\pi = \pi^*$
$end\ if$
$end\ for$
$return\ \pi\ and\ f(\pi)$

In the stochastic variant given in Algorithm 4, jobs are randomly chosen from solutions to make insertions. In Algorithm 4, job π_k at position k is randomly chosen from the solution π without repetition, and partial solution π_P is obtained. Then, job π_k is inserted into all possible positions of the partial solution π_P. When the best-improving insertion position is found, job π_k is inserted into that position, and a complete solution π^* is obtained. These steps are repeated for all jobs. If an improvement is found, the local search is rerun again until no better solution is obtained.

Algorithm 4: First improvement insertion neighborhood(π)

$for\ i = 1\ to\ n\ do$
$\pi_P = Remove\ job\ \pi_k\ from\ solution\ \pi\ randomly\ and\ without\ repetition$
$\pi^* = InsertJobInBestPosition(\pi_P,\ \pi_k)$
$if\ (f(\pi^*) < f(\pi))\ then\ do$
  $\pi = \pi^*$
$end\ if$
$end\ for$
$return\ \pi\ and\ f(\pi)$

Recently, a new IG_{ALL} algorithm has been presented in the literature [5] with excellent results for the PFSP with makespan minimization. The difference between IG_{ALL} and IG_{RS} is that IG_{ALL} applies an additional local search to partial solutions after destruction, which substantially enhances solution quality. In the IG_{RS} algorithm, local search is applied to the complete solution after the construction phase to improve the current candidate solution whereas, in IG_{ALL} algorithm, local search is applied to a partial solution after destruction phase. This idea is applied in heuristic algorithms by Reference [39].

They study on vehicle routing problem and apply local search on the routes in the construction phase. Applying local search to the partial solution is more advantageous in terms of computational time and providing different search directions. Due to having a partial solution, a local search is applied to the smaller size of the complete solution so that the search procedure can be conducted quickly. Another difference between IG_{RS} and IG_{ALL} is due to the fact that the initial solution is constructed by FRB5 heuristic. The pseudo code of IG_{ALL} algorithm is presented in Algorithm 5.

Algorithm 5: IG_{ALL} algorithm

$\pi = \text{FRB5}$
$\pi^{best} = \pi$
While $(NotTermination)$ *do*
    $\pi_D = Destruction(\pi)$
    $\pi_P = LocalSearchToPartialSolution(\pi_P)$ $//Algorithm\ 4$
    $\pi^1 = Construction(\pi_P, \pi_D)$
    $\pi^1 = LocalSearchToCompleteSolution(\pi^1)$ $//Algorithm\ 4$
    $if\ f(\pi^1) \leq f(\pi) then\ do$
        $\pi = \pi^1$
        $if\ f(\pi^1) < f(\pi^{best}) then\ do$
            $\pi^{best} = \pi^1$
        $endif$
        $else\ if\ (r < exp\{-(f(\pi^1) - f(\pi))/T\})$
            $\pi = \pi^1$
        $endif$
    $endif$
endwhile
return π^{best} *and* $f(\pi^{best})$
endprocedure

Note that Algorithm 3 is used in the FRB5 heuristic in order to construct the initial solution with a single run due to its deterministic property. In both algorithms, Algorithm 4 is employed in applying local search to both partial and complete solutions.

3.3. Variable Block Insertion Algorithm

In this paper, we propose a VBIH algorithm as follows. The VBIH algorithm employs the FRB5 heuristic as an initial solution. It has a minimum block size (b_{min}), and a maximum block size (b_{max}). It removes a block of jobs (π_b) with size b from the current solution and obtains a partial solution (π_P). Similar to the IG_{ALL} algorithm, it applies the local search in Algorithm 4 to the partial solution. Then, it makes a number, $n - b + 1$, of block insertion moves sequentially in the partial solution. It chooses the best one amongst those solutions from block insertion moves. Well-known RIS local search in the literature is applied to the complete solution found after block insertion moves. If the new solution obtained after the local search is better than or equal to the current solution, it replaces the current solution. As long as it improves, it retains the same block size (*i.e.*, $b = b$). Otherwise, the block size is incremented by one (*i.e.*, $b = b + 1$) and a simulated annealing-based acceptance criterion, similar to IG_{RS} and IG_{ALL} algorithms, is employed to accept the new solution to escape from local minima. This process is repeated until the block size reaches its maximum limit (*i.e.*, $b \leq b_{max}$). The outline of the VBIH algorithm is given in Algorithm 6. Note that π^R is the reference sequence; tP is temperature parameter for the acceptance criterion and r is a uniform random number between 0 and 1.

Algorithm 6: VBIH algorithm

$\pi = FRB5$
$\pi^{best} = \pi$
$\pi^{R} = \pi^{best}$
$while\ (NotTermination)$
  $b = b_{min} = 2$
  do
    $\pi_b = Remove\ block\ \pi_b\ from\ \pi$
    $\pi_P = LocalSearchToPartialSolution(\pi_P)$ $//Algorithm$ 4
    $\pi^1 = InsertBlockInBestPosition(\pi_P, \pi_b)$
    $\pi^1 = RISLocalSearchToCompleteSolution(\pi^1)$ $//Algorithm$ 5
    $if\ (f(\pi^1) < f(\pi))\ then\ do$
        $\pi = \pi^1$
        $b = b$
        $if\ (f(\pi^1) < f(\pi_{best}))then\ do$
            $\pi^{best} = \pi^1$
            $\pi^{R} = \pi^{best}$
        $endif$
    $else$
        $b = b + 1$
        $if\ (r < exp\left\{-(f(\pi^1) - f(\pi))/T\right\})$
            $\pi = \pi^1$
        $endif$
    $endif$
  $while((b \leq b_{max})$
$endwhile$
$return\ \pi^{best}\ and\ f(\pi^{best})$

To explain the block insertion procedure, we give the following example. Suppose that we are given a current solution $\pi = \{1, 2, 3, 4, 5\}$. Furthermore, assume that the block size is $b = 2$. Let's randomly choose a block $\pi_b = \{2, 5\}$, thus ending up with a partial solution, $\pi_p = \{1, 3, 4\}$. After applying local search to the partial solution π_p, suppose that we have a partial solution $\pi_p = \{3, 1, 4\}$. Now, the block π_b is inserted into four positions as follows: $\pi^1 = \{2, 5, 3, 1, 4\}$, $\pi^2 = \{3, 2, 5, 1, 4\}$, $\pi^3 = \{3, 1, 2, 5, 4\}$ and $\pi^4 = \{3, 1, 4, 2, 5\}$. Among these four solutions, the best one will be chosen as a final solution.

Regarding the local search algorithm that will be applied only to complete solutions, we use a well-known referenced insertion scheme local search, RIS [8,40]. RIS is guided by a reference solution π^R, which is the best solution obtained so far during the search process. For instance, if the reference solution is given by $\pi^R = \{3, 5, 1, 4, 2\}$ and the current solution by $\pi = \{1,2,3,4,5\}$. The reference solution π^R implies that job 3 in the current solution π might not be in a proper position. For this reason, the RIS local search first removes job 3 from the current solution π and inserts it into all possible slots of the partial solution π_P. A new solution with the best insertion slot is replaced by the current solution, and the iteration counter is reset to one if any improvement occurs. Otherwise, the iteration counter is incremented by one. Then, it removes job 5 from the current solution π and inserts it into all possible positions of the partial solution π_P. This procedure is repeated until the iteration counter is greater than the number of jobs n, and a new solution is obtained. The pseudo-code of the RIS local search is given in Algorithm 7.

After the local search phase, it should be decided if the new solution is accepted as the incumbent solution for the next iteration. A simple simulated annealing-type of acceptance criterion is used with a constant temperature similar to the IG_{RS} and IG_{ALL} algorithms. Note that Taillard's speed-ups are employed wherever possible in our code.

Algorithm 7: Referenced insertion neighborhood(π)

```
Count = 1
pos = 1
π^R = π^best
while (Count ≤ n) do
    k = 1
    while (π_k != π^R_Pos) k = k + 1; endwhile        //Find job π_k at position pos in π^R
    pos = pos + 1
    if(pos = n + 1) then
        pos = 1
    end if
    π_P = remove π_k from π
    π* = InsertJobInBestPosition(π_P, π_k)
    if (f(π*) < f(π)) then do
        π = π*
        Count = 1
    end
        Count = Count + 1
    end if
endwhile
return π and f(π)
```

4. Design of Experiment for Parameter Tuning

In this section, we present a Design of Experiments (DOE) approach [41] for parameter settings of the VBIH algorithm. In order to carry out experiments, we generate random instances with the method proposed in [9]. In other words, random instances are generated for each combination of $n \in \{100, 200, 300, 400, 500, 600, 700, 800\}$ and $m \in \{20, 40, 60\}$. Five instances are generated for each job and machine combination. Ultimately, we obtained 1200 instances in total. We consider three parameters in the DOE approach. These are maximum block size $(bMax)$, temperature adjustment parameter (τP), and the decision of whether or not to implement the local search to the partial solution after removal of a block of jobs. We have taken the maximum block size with seven levels as $bMax \in (2, 3, 4, 5, 6, 7, 8)$; the temperature adjustment parameter with ten levels as $\tau P \in (0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5)$; and the decision on the local search to partial solutions as $pL \in (1, 2)$. Note that $pL = 1$ means that the local search is applied to partial solutions whereas $pL = 2$ does not apply the local search to partial solutions. In the design of VBIH algorithm, there are $7 \times 5 \times 2 = 70$ algorithm configurations, i.e., treatments. The VBIH algorithm is coded in C++ programming language on Microsoft Visual Studio 2013, and a full factorial design of experiments is carried out for each algorithm on a Core i5, 3.40 GHz, 8 GB RAM computer. Each instance is run for 70 treatments with a maximum CPU time $T_{max} = 10 \times n \times m$ milliseconds. Note that it took 18 days to run the full factorial design. We calculate the relative percent deviation (RPD) for each instance-treatment pair as follows:

$$RPD = \left(\frac{CMAX_i - CMAX_{min}}{CMAX_{min}}\right) \times 100 \tag{18}$$

where $CMAX_i$ is the makespan value generated by the VBIH algorithm in each treatment and $CMAX_{min}$ is the minimum makespan value found amongst 70 treatments. For each job size-treatment pair, the average RPD value is calculated by taking the average of five instances in each job size. Then, the response variable (ARPD) of each treatment is obtained by averaging these RPD values of all job sizes. After determining the ARPD values for each treatment as mentioned above, the main effects plots of the parameters are analyzed and given in Figure 3.

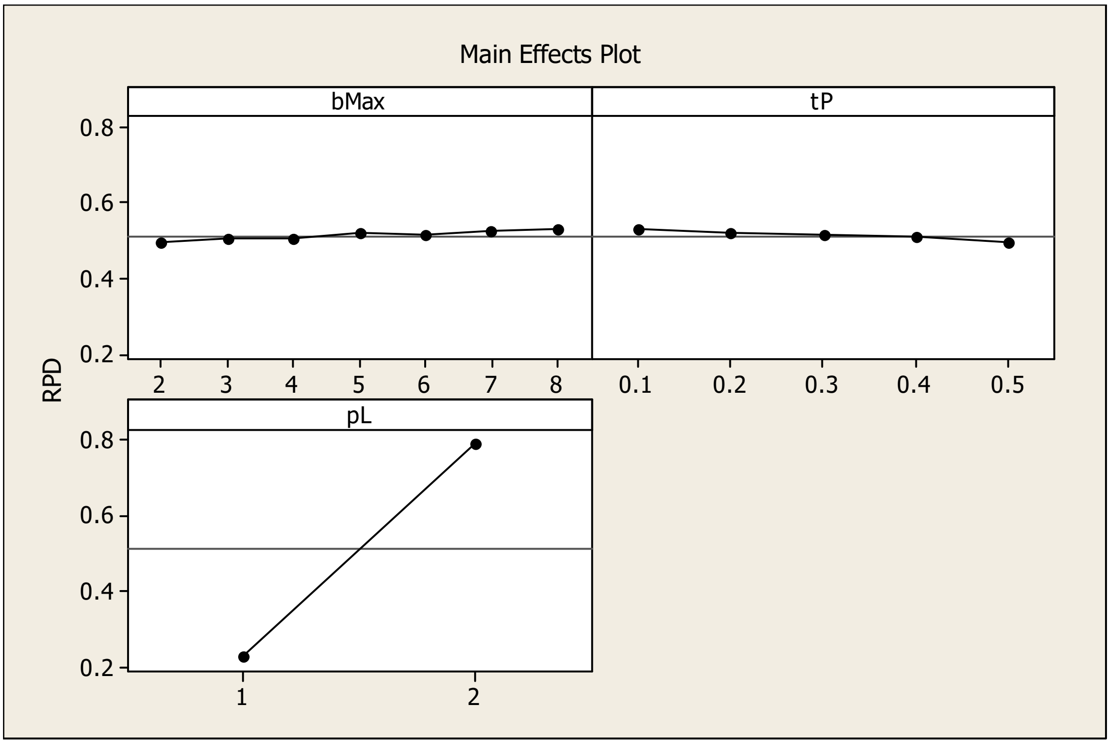

Figure 3. Main effects plot for parameters of VBIH.

As it can be seen from Figure 3, the following parameters have better ARPD values than the others: $bMax = 2$, $\tau P = 0.5$, and $pL = 1$. Furthermore, in order to see whether or not there is an interaction effect between parameters, an ANOVA analysis is also given in Table 2.

Table 2. ANOVA results for parameters of VBIH.

Source	*DF*	*Seq SS*	*Adj SS*	*Adj MS*	*F*	*p*-Value
bMax	6	0.0086	0.0086	0.0014	33.370	0.000
tP	4	0.0090	0.0090	0.0022	52.080	0.000
pL	1	5.5441	5.5441	5.5441	129,096.720	0.000
$bMax \times tP$	24	0.0010	0.0010	0.0000	0.990	0.505
$bMax \times pL$	6	0.0025	0.0025	0.0004	9.830	0.000
$tP \times pL$	4	0.0090	0.0090	0.0022	52.100	0.000
Error	24	0.0010	0.0010	0.0000		
Total	69	5.5752				

Table 2 indicates that *bMax*, *tP*, and *pL* were statistically significant since higher magnitude of *F* values and *p*-values of parameter interaction effects are less than the significance level $\alpha = 0.05$. High magnitude of *F* value for *pL* also suggest that applying local search to partial solutions has a significant impact on the solution quality as mentioned in [5]. In terms of interaction effects, it can be observed that $bMax \times tP$ interaction is not significant because the *p*-value is much higher than the significance level $\alpha = 0.05$. However, $bMax \times pL$ and $tP \times pL$ interactions were significant since their p values are less than the significance level $\alpha = 0.05$. The interaction effects plot for $bMax \times pL$ is given in Figure 4.

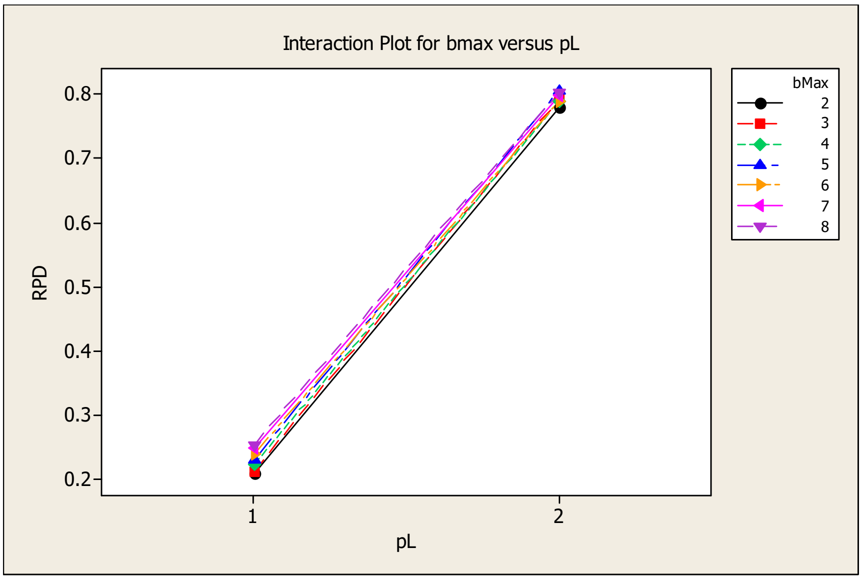

Figure 4. Interaction plot for *bMax* versus *pL*.

Figure 4 indicates that maximum block size should be taken as $bMax = 2$ and local search to the partial solution should be applied. Since $tP \times pL$ interaction is also significant, we provide the interaction plot in Figure 5.

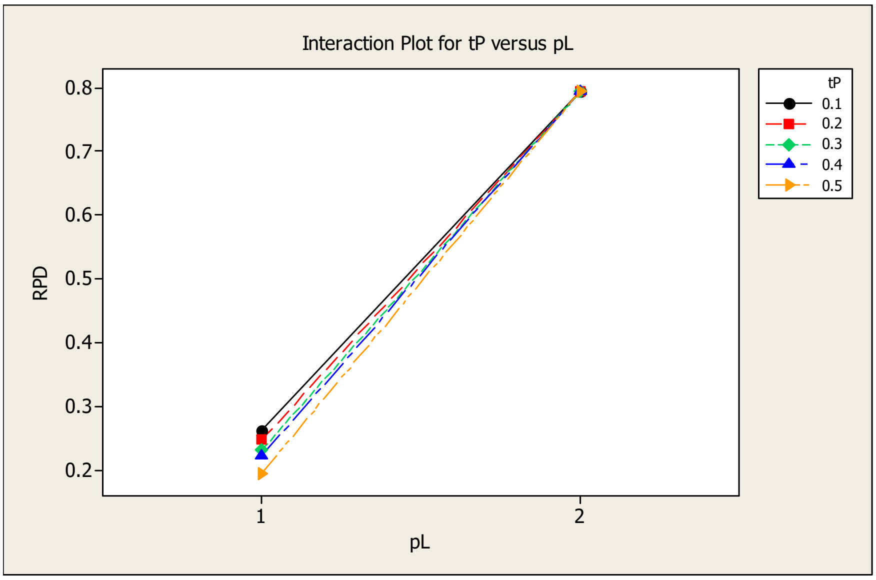

Figure 5. Interaction plot for *tP* versus *pL*.

Figure 5 also suggests that *tP* and *pL* parameters should be taken as $\tau P = 0.5$ and $pL = 1$. Ultimately, we set the parameters of VBIH algorithm as follows: $bMax = 2$, $\tau P = 0.5$, and $pL = 1$.

5. Computational Results

In this section, the computational results for the small and large set of VRF benchmark sets are provided. MIP and CP models were written in OPL and run on the IBM ILOG CPLEX 12.8 software suite, while all the heuristic algorithms were being written in Visual C++ 13 and carried out on an Intel Core i5, 3.40 GHz, 8 GB RAM computer. The proposed VBIH algorithm is compared to IG_{RS} and IG_{ALL} algorithms. In addition, the results of these algorithms are obtained without the Taillard's speed up method, and they are denoted as IG_{RS}*, IG_{ALL}* and VBIH*. Regarding parameters of them with, destruction size ds, and temperature adjustment factor, tP are taken as $ds = 4$ and $tP = 0.4$ for IG_{RS} and IG_{RS}* as suggested in [4]. They are taken as $ds = 2$ and $tP = 0.7$ for IG_{ALL} and IG_{ALL}* as indicated in [5]. As explained in the previous section DOE is conducted for the VBIH algorithm and its parameters are determined as follows: $bMax = 2$, $\tau P = 0.5$, and $pL = 1$, which are also used for the VBIH* algorithm.

5.1. Small VRF Instances

5.1.1. MIP Versus CP

Computational results are given in Table 3 for each combination, giving a total of 240 small VRF instances. For each combination, the table summarizes the number of optimal solutions (nOpt) found for ten instances of each job-machine combination ($n \times m$), the average relative percent deviation (ARPD%) from the upper bounds given in [9], the average CPU time for its ten instances, and the optimality gap percentage (GAP%) on termination, which means the gap between best lower and best upper bound. The maximum CPU time is restricted to an hour (3600 s). The result of CP and MIP models are compared for job sizes 10 and 20. While MIP model can find solutions for very small sized instances (10 jobs) in a shorter time than CP model, it becomes hard for MIP to solve large sized problems (20 jobs and more). Both models cannot always find optimal solutions when the machine size becomes greater than 5, but the MIP model has larger gaps than the CP model. The results show that CP is more efficient than MIP on PFSP, except for very small-sized instances. The results of the remaining instances are obtained only from the CP model because of very large gaps by MIP model. CP model always captures optimal solutions when the machine number is five regardless of the number of jobs. Besides, CP can find optimal solutions in some of the instances when the machine size is 10. Overall, within the time limit, the CP model verifies optimality for 108 out of 240 instances.

Table 3. MIP and CP results for VRF small benchmarks with 3600 s time limit (The number in bold shows the total optimal solutions).

$n \times m$	CP				MIP			
	***n*Opt**	**ARPD**	**CPU**	**GAP**	**nOpt**	**RPD**	**CPU**	**GAP**
10×5	10	0	14.03	0	10	0	2.68	0
10×10	10	0	102.13	0	10	0	4.35	0
10×15	10	0	256.45	0	10	0	5.68	0
10×20	10	0	452.79	0	10	0	9.59	0
20×5	10	0	2.49	0	0	0.58	3600.18	0.37
20×10	6	0.11	2250.09	0.03	0	2.24	3600.51	0.32
20×15	0	0.53	3600.05	0.13	0	2.54	3600.06	0.29
20×20	0	0.48	3600.07	0.17	40	2.61	3600.06	0.25
30×5	10	0	5.82	0	Na	Na	Na	Na
30×10	2	0.47	3191.89	0.05	Na	Na	Na	Na
30×15	0	1.29	3600.14	0.11	Na	Na	Na	Na
30×20	0	1.63	3600.13	0.15	Na	Na	Na	Na
40×5	10	0	15.03	0	Na	Na	Na	Na
40×10	3	0.22	3113.36	0.03	Na	Na	Na	Na
40×15	0	2.16	3600.10	0.10	Na	Na	Na	Na
40×20	0	2.11	3600.16	0.13	Na	Na	Na	Na
50×5	10	0	11.64	0	Na	Na	Na	Na
50×10	3	0.19	2939.96	0.02	Na	Na	Na	Na
50×15	0	2.28	3600.22	0.08	Na	Na	Na	Na
50×20	0	2.73	3600.22	0.12	Na	Na	Na	Na
60×15	10	0	6.44	0	Na	Na	Na	Na
60×10	4	0.19	3158.95	0.01	Na	Na	Na	Na
60×15	0	1.98	3600.19	0.07	Na	Na	Na	Na
60×20	0	2.82	3600.29	0.10	Na	Na	Na	Na
Overall Avg.	**108**	0.80	2146.78	0.05	40	2.61	3600.06	0.25

5.1.2. Comparison of Heuristic Algorithms with Exact Solutions

In order to compare performances of heuristic algorithms with CP exact method, we run all algorithms for five independent replications with different seed numbers. Relative percent deviation values from upper bounds for ten different instances of each job-machine combinations are calculated as follows:

$$RPD = \frac{(M - M_{UB}) \times 100}{M_{UB}} \tag{19}$$

where M is the makespan value generated by any heuristic; and M_{UB} is the upper bound provided in [9]. Note that, for each instance, we record the average RFD of five replications for statistical analysis purposes, especially, for interval graphs. The solutions of the CP model are limited to 3600 s and its average CPU times are given in Table 4. IG_{ALL}, IG_{RS}, and VBIH algorithms are run for five replications with three different time limits 15, 30, and $45 \times n \times m$. As expected, the performance of these algorithms is much better than those by CP exact model, and they improve the upper bounds provided in [9], which means that the proposed algorithm and other IG algorithms can find good (optimal in some cases) solutions in a very short time. As the solution time increases, the solution quality of VBIH algorithm increases and according to the RPD, it gives the best solutions amongst all other algorithms. It should be noted that the VBIH algorithm further improves 64 out 240 upper bounds for small VRF instances within a very short time.

Table 4. Comparison of ARPD of all algorithms for small VRF instances.

Instance	CP	$15 \times n \times m$			$30 \times n \times m$			$45 \times n \times m$		
		IG_{RS}	IG_{ALL}	VBIH	IG_{RS}	IG_{ALL}	VBIH	IG_{RS}	IG_{ALL}	VBIH
10×5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10×10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10×15	0.00	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.00	0.02
10×20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20×5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20×10	0.11	0.04	0.00	0.04	0.03	0.00	0.04	0.02	0.00	0.04
20×15	0.53	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20×20	0.48	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30×5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30×10	0.47	0.06	0.04	0.05	0.01	0.03	0.01	0.01	0.03	−0.01
30×15	1.29	0.03	0.02	0.03	0.02	−0.02	0.02	0.02	−0.02	0.02
30×20	1.63	0.02	0.00	0.03	0.02	0.00	0.02	0.02	0.00	0.02
40×5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40×10	0.22	0.06	0.02	0.03	0.02	0.01	−0.01	0.00	0.00	−0.01
40×15	2.16	0.09	0.05	0.04	0.04	0.02	−0.02	−0.01	−0.05	−0.05
40×20	2.11	0.10	−0.08	−0.04	0.04	−0.08	−0.05	−0.01	−0.08	−0.07
50×5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50×10	0.19	0.16	0.14	0.04	0.11	0.11	0.00	0.08	0.08	−0.03
50×15	2.28	0.24	0.18	0.10	0.15	0.14	0.05	0.10	0.09	0.02
50×20	2.73	0.17	0.02	0.00	0.07	−0.08	−0.10	0.04	−0.11	−0.10
60×5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
60×10	0.19	0.07	0.11	−0.01	−0.04	0.08	−0.03	−0.06	0.05	−0.05
60×15	1.98	0.21	0.09	0.10	0.12	0.06	0.01	0.08	0.06	−0.04
60×20	2.81	0.20	0.01	0.00	0.03	−0.07	−0.12	−0.03	−0.08	−0.17
Avg.	0.80	0.06	0.02	0.02	0.03	0.01	−0.01	0.01	0.00	−0.02

5.2. Large VRF Instances

Note that both IG_{ALL} and VBIH algorithms employ the FRB5 heuristic for constructing initial solution whereas IG_{RS} uses the traditional NEH heuristic. For the large VRF instances, Table 5 summarizes the ARPD values of heuristic methods such as NEH, NEH without speed-up, denoted as NEH*, and extended NEH heuristic with a local search on partial solutions denoted as FRB5.

Table 5. Comparison of ARPD and computation time (CPU) for constructive heuristic methods (The number in bold shows better results).

Instance	NEH		NEH *		FRB5	
	ARPD	**CPU(s)**	**ARPD**	**CPU(s)**	**ARPD**	**CPU(s)**
100 × 20	5.82	0.00	5.82	0.01	2.45	0.10
100 × 40	5.30	0.00	5.30	0.03	2.57	0.21
100 × 60	4.89	0.00	4.89	0.05	2.19	0.32
200 × 20	4.15	0.00	4.15	0.10	1.42	0.89
200 × 40	4.81	0.01	4.81	0.23	1.67	1.91
200 × 60	4.48	0.01	4.48	0.39	1.56	2.73
300 × 20	3.17	0.01	3.17	0.33	0.80	2.75
300 × 40	4.05	0.02	4.05	0.79	1.07	6.45
300 × 60	3.94	0.03	3.94	1.31	1.23	9.85
400 × 20	2.44	0.01	2.44	0.80	0.50	6.27
400 × 40	3.80	0.03	3.80	1.91	0.82	15.83
400 × 60	3.42	0.04	3.42	3.14	0.75	24.39
500 × 20	2.06	0.02	2.06	1.53	0.43	12.10
500 × 40	3.17	0.04	3.17	3.75	0.63	31.73
500 × 60	3.27	0.06	3.27	6.05	0.57	47.97
600 × 20	1.70	0.03	1.70	2.60	0.24	20.76
600 × 40	2.96	0.06	2.96	6.34	0.53	54.97
600 × 60	2.97	0.09	2.97	10.31	0.37	82.27
700 × 20	1.42	0.04	1.42	4.13	0.25	31.50
700 × 40	2.80	0.08	2.80	10.06	0.26	84.38
700 × 60	2.66	0.13	2.66	17.22	0.32	249.99
800 × 20	1.35	0.04	1.35	6.06	0.21	42.31
800 × 40	2.45	0.10	2.45	15.48	0.24	125.13
800 × 60	2.74	0.16	2.74	26.17	0.31	195.41
Avg	3.33	0.04	3.33	4.95	**0.89**	43.76

As shown in Table 5, NEH is very fast with 0.04 s on overall average CPU time. However, its overall average of ARPD is 3.33%. Although FRB5 heuristic is computationally very expensive, which is 43.76 s on overall average CPU time, its average ARPD is only 0.89% from the upper bounds. It is obvious from Table 5 that FRB5 heuristic is substantially better than NEH with a very large margin at the expense of increased CPU time. It is interesting to observe the CPU time performance of the NEH heuristic without the speed-up method of Taillard. Table 5 clearly indicates that the Taillard's speed-up method is substantially effective since the overall average CPU time is jumped from 0.04 s to 4.95 s without the speed-up method of Taillard. In addition to the above, we present the interval graph of both constructive heuristics in Figure 6 in order for justification. Figure 6 indicates that differences in ARPDs are significantly meaningful on the behalf of FRB5 heuristic since their confidence intervals do not coincide.

5.3. Computational Results of Metaheuristics

In this section, the performance of VBIH algorithm is compared to the best-performing algorithms, IG_{RS} and IG_{ALL}, from the literature. All algorithms are run five replications to solve the large VRF instances. In Table 6, we present average, minimum and maximum ARPD values for the CPU time limit $T_{max} = 15 \times n \times m$ milliseconds.

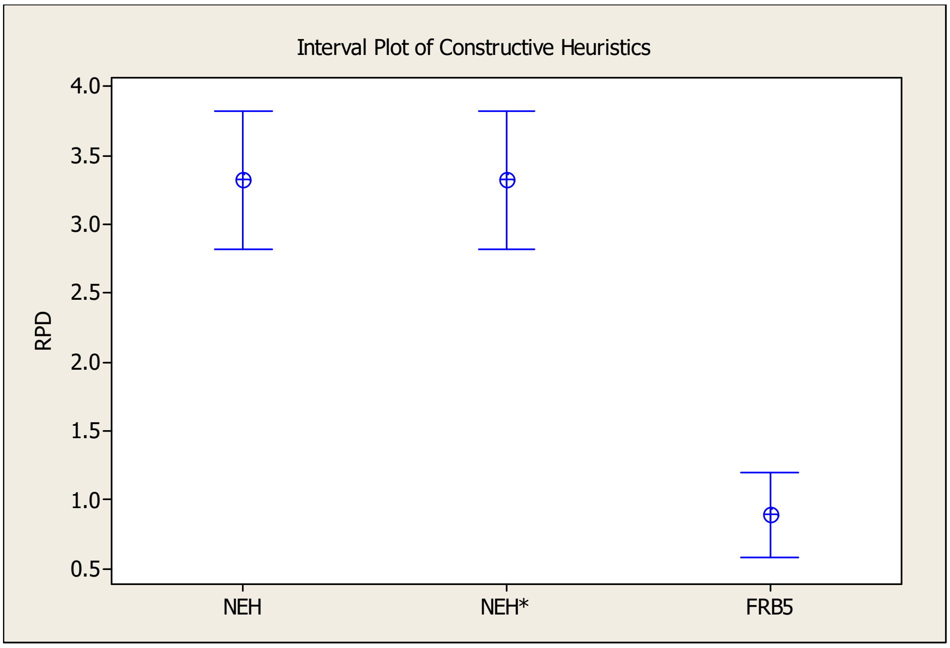

Figure 6. Interval plot for small VRF instances.

Table 6. Computational results of algorithms with $T_{max} = 15 \times n \times m$ milliseconds (The bolds show better results).

Instance	IG_{RS}			IG_{ALL}			VBIH		
	Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max
100 × 20	0.45	0.13	0.74	0.12	−0.07	0.33	0.00	−0.21	0.23
100 × 40	0.56	0.26	0.90	0.28	0.04	0.49	0.13	−0.09	0.37
100 × 60	0.50	0.22	0.78	0.23	0.02	0.42	0.27	0.05	0.54
200 × 20	0.42	0.24	0.61	0.19	0.04	0.35	0.03	−0.14	0.17
200 × 40	0.47	0.25	0.68	0.14	−0.01	0.31	0.01	−0.21	0.24
200 × 60	0.46	0.24	0.65	0.17	−0.01	0.37	0.05	−0.15	0.22
300 × 20	0.22	0.06	0.35	0.10	−0.03	0.21	−0.03	−0.17	0.11
300 × 40	0.35	0.15	0.56	0.04	−0.16	0.25	−0.18	−0.35	−0.02
300 × 60	0.36	0.16	0.56	0.12	−0.06	0.27	−0.03	−0.20	0.15
400 × 20	0.20	0.11	0.33	0.09	0.01	0.18	0.03	−0.03	0.10
400 × 40	0.31	0.12	0.50	0.01	−0.11	0.14	−0.17	−0.32	−0.03
400 × 60	0.27	0.08	0.46	−0.02	−0.17	0.12	−0.16	−0.27	−0.05
500 × 20	0.15	0.06	0.26	0.12	0.07	0.18	0.03	−0.05	0.12
500 × 40	0.29	0.12	0.45	0.00	−0.10	0.11	−0.19	−0.30	−0.07
500 × 60	0.33	0.15	0.51	−0.06	−0.20	0.08	−0.19	−0.31	−0.06
600 × 20	0.11	0.03	0.18	0.02	−0.03	0.07	0.01	−0.05	0.06
600 × 40	0.38	0.23	0.54	0.03	−0.07	0.13	−0.05	−0.17	0.06
600 × 60	0.30	0.12	0.50	−0.05	−0.18	0.05	−0.13	−0.23	−0.04
700 × 20	0.11	0.05	0.18	0.04	−0.01	0.08	0.03	−0.03	0.08
700 × 40	0.24	0.13	0.37	−0.11	−0.20	0.00	−0.21	−0.28	−0.12
700 × 60	0.26	0.09	0.46	−0.05	−0.15	0.04	−0.13	−0.24	−0.03
800 × 20	0.07	0.02	0.14	0.06	0.02	0.12	0.01	−0.04	0.05
800 × 40	0.22	0.09	0.36	−0.06	−0.14	0.02	−0.25	−0.33	−0.17
800 × 60	0.40	0.25	0.57	0.02	−0.04	0.08	−0.19	−0.29	−0.10
Avg	0.31	0.14	0.48	0.06	−0.06	0.18	**−0.05**	**−0.18**	**0.08**

As seen in Table 6, VBIH generated better Avg, Min and Max RPD values on the overall average. On overall average, it was able to further improve the upper bounds up to −0.05%; its best overall performance was −0.18% indicating that 0.18% of 240 instances are further improved and its worst-case performance was 0.08%. In order to see if differences in ARPDs are statistically significant, we provide the 95% confidence interval plot of algorithms in Figure 7, where we can observe that differences in ARPD values are statistically significant on the behalf of VBIH against IG_{RS} and IG_{ALL} algorithms because their confidence intervals do not coincide.

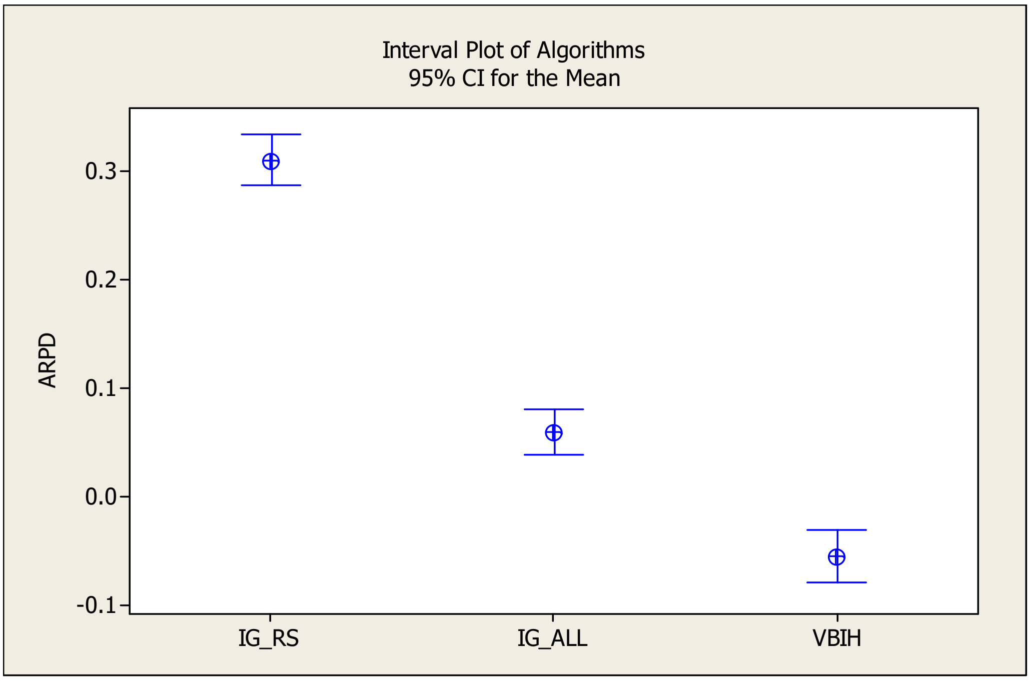

Figure 7. Interval plot at the 95% confidence level for large VRF instances.

Computational results for Avg, Min and Max ARPD values with the CPU time limit $T_{max} = 30 \times n \times m$ milliseconds are given in Table 7. As seen in Table 7, VBIH was able to generate better Avg, Min and Max ARPD values on the overall average. On overall average, it was able to further improve the upper bounds by −0.11% in Avg value, −0.24% of upper bounds are further improved on Min value and its worst-case performance was 0.02%. However, as CPU times increased, the performance of IG_{ALL} algorithm was also remarkable. Briefly, both VBIH and IG_{ALL} outperformed IG_{RS} in almost each problem set.

In order to see if these results are statistically significant, we provide the 95% confidence interval plot of algorithms in Figure 8, where we can observe that differences in ARPD values are statistically significant on the behalf of both VBIH and IG_{ALL} algorithms against IG_{RS} algorithm because their confidence intervals do not coincide with IG_{RS}. In other words, VBIH and IG_{ALL} algorithms were statistically equivalent but significantly superior to IG_{RS}.

Table 7. Computational results of algorithms with $T_{max} = 30 \times n \times m$ milliseconds (The bolds show better results).

$n \times m$	IG_{RS}			IG_{ALL}			VBIH		
	Avg.	**Min**	**Max**	**Avg.**	**Min**	**Max**	**Avg.**	**Min**	**Max**
100 × 20	0.25	−0.02	0.54	0.03	−0.11	0.16	−0.05	−0.25	0.16
100 × 40	0.38	0.08	0.68	0.05	−0.14	0.23	0.07	−0.15	0.33
100 × 60	0.36	0.13	0.63	0.05	−0.17	0.23	0.21	−0.02	0.51
200 × 20	0.28	0.12	0.45	0.07	−0.05	0.22	0.00	−0.16	0.14
200 × 40	0.30	0.06	0.51	−0.08	−0.25	0.08	−0.04	−0.25	0.16
200 × 60	0.26	0.05	0.51	−0.04	−0.19	0.13	0.02	−0.17	0.19
300 × 20	0.12	−0.01	0.23	0.01	−0.10	0.14	−0.06	−0.21	0.08
300 × 40	0.17	−0.03	0.41	−0.22	−0.37	−0.04	−0.23	−0.39	−0.07
300 × 60	0.18	−0.03	0.42	−0.08	−0.25	0.12	−0.09	−0.24	0.07
400 × 20	0.12	0.04	0.19	0.03	−0.04	0.09	0.01	−0.06	0.09
400 × 40	0.16	−0.03	0.37	−0.20	−0.38	−0.07	−0.22	−0.36	−0.08
400 × 60	0.08	−0.11	0.24	−0.22	−0.37	−0.07	−0.20	−0.31	−0.11
500 × 20	0.11	0.02	0.20	0.07	0.01	0.13	0.02	−0.06	0.10
500 × 40	0.13	−0.05	0.32	−0.16	−0.26	−0.06	−0.24	−0.36	−0.12
500 × 60	0.15	−0.03	0.32	−0.22	−0.35	−0.09	−0.23	−0.35	−0.10
600 × 20	0.07	−0.02	0.15	−0.01	−0.06	0.04	−0.02	−0.07	0.03
600 × 40	0.20	0.04	0.36	−0.11	−0.19	−0.02	−0.19	−0.29	−0.07
600 × 60	0.13	−0.03	0.32	−0.23	−0.37	−0.11	−0.26	−0.37	−0.15
700 × 20	0.08	0.01	0.16	0.02	−0.03	0.06	−0.01	−0.07	0.03
700 × 40	0.09	−0.01	0.19	−0.27	−0.38	−0.15	−0.34	−0.42	−0.27
700 × 60	0.07	−0.11	0.23	−0.21	−0.28	−0.13	−0.28	−0.39	−0.19
800 × 20	0.04	−0.01	0.09	0.02	−0.01	0.05	0.00	−0.04	0.04
800 × 40	0.07	−0.07	0.21	−0.20	−0.30	−0.11	−0.28	−0.35	−0.21
800 × 60	0.22	0.10	0.40	−0.13	−0.22	−0.04	−0.23	−0.32	−0.13
Avg	0.17	0.00	0.34	−0.08	−0.20	0.03	**−0.11**	**−0.24**	**0.02**

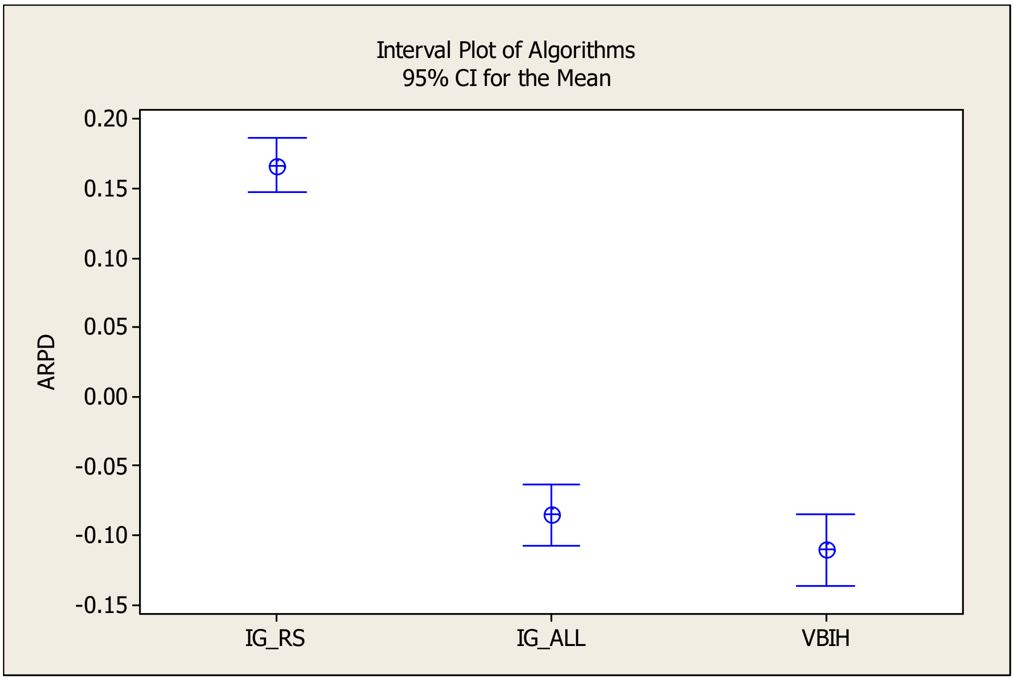

Figure 8. Interval plot at the 95% confidence level for large VRF instances.

Computational results for average, minimum and maximum RPD values with the CPU time limit $T_{max} = 45 \times n \times m$ milliseconds are given in Table 8, where VBIH outperformed IG_{RS} and IG_{ALL} algorithms with respect to average, minimum and maximum RPD values on the overall average. On overall average, it was able to further improve the upper bounds by −0.25% on the average value, −0.36% on the minimum value, and its worst-case performance was −0.13%. These statistics indicate that VBIH generated much better results than both the IG_{RS} and IG_{ALL} algorithms.

Table 8. Computational results of algorithms with $T_{max} = 45 \times n \times m$ milliseconds (The bolds show better results).

$n \times m$	IG_{RS}			IG_{ALL}			VBIH		
	Avg.	**Min**	**Max**	**Avg.**	**Min**	**Max**	**Avg.**	**Min**	**Max**
100 × 20	0.13	−0.14	0.39	−0.04	−0.21	0.1	−0.25	−0.44	−0.03
100 × 40	0.29	0.02	0.59	−0.05	−0.25	0.13	−0.18	−0.35	−0.01
100 × 60	0.26	0.03	0.48	−0.03	−0.28	0.17	−0.02	−0.17	0.19
200 × 20	0.21	0.05	0.37	0	−0.14	0.12	−0.12	−0.27	0.03
200 × 40	0.21	0.01	0.4	−0.2	−0.36	−0.03	−0.3	−0.53	−0.07
200 × 60	0.14	−0.07	0.37	−0.14	−0.3	0.02	−0.27	−0.43	−0.1
300 × 20	0.07	−0.06	0.17	−0.04	−0.18	0.1	−0.15	−0.26	−0.05
300 × 40	0.06	−0.13	0.27	−0.33	−0.47	−0.17	−0.45	−0.56	−0.28
300 × 60	0.08	−0.14	0.34	−0.24	−0.4	−0.04	−0.32	−0.47	−0.17
400 × 20	0.09	0	0.17	−0.03	−0.12	0.02	−0.05	−0.12	0.01
400 × 40	0.09	−0.09	0.3	−0.44	−0.57	−0.3	−0.41	−0.52	−0.28
400 × 60	−0.03	−0.23	0.16	−0.48	−0.64	−0.31	−0.41	−0.52	−0.32
500 × 20	0.07	−0.02	0.18	0.02	−0.06	0.08	−0.04	−0.11	0.06
500 × 40	0.04	−0.16	0.21	−0.41	−0.53	−0.29	−0.42	−0.5	−0.29
500 × 60	0.02	−0.14	0.17	−0.44	−0.56	−0.3	−0.41	−0.54	−0.29
600 × 20	0.04	−0.04	0.13	−0.04	−0.08	0.01	−0.05	−0.08	−0.01
600 × 40	0.11	−0.05	0.29	−0.32	−0.41	−0.21	−0.27	−0.39	−0.15
600 × 60	0.03	−0.12	0.22	−0.45	−0.6	−0.33	−0.35	−0.44	−0.23
700 × 20	0.06	−0.02	0.14	0	−0.05	0.05	−0.03	−0.08	0.02
700 × 40	0.01	−0.11	0.13	−0.36	−0.48	−0.24	−0.42	−0.5	−0.35
700 × 60	−0.01	−0.2	0.16	−0.3	−0.4	−0.22	−0.37	−0.48	−0.25
800 × 20	0.02	−0.04	0.07	0.01	−0.03	0.04	−0.01	−0.06	0.03
800 × 40	−0.01	−0.15	0.12	−0.27	−0.36	−0.17	−0.36	−0.43	−0.29
800 × 60	0.13	0	0.31	−0.21	−0.3	−0.14	−0.32	−0.4	−0.22
Average	0.09	−0.07	0.26	−0.20	−0.32	−0.08	**−0.25**	**−0.36**	**−0.13**

In order to see if these results are statistically significant, we provide the 95% confidence interval plot of algorithms in Figure 9, where we can observe that differences in ARPD values are statistically significant on the behalf of VBIH algorithm against both IG_{RS} and IG_{ALL} algorithms because their confidence intervals do not coincide. In other words, VBIH algorithm was statistically superior to both IG_{RS} and IG_{ALL} algorithm.

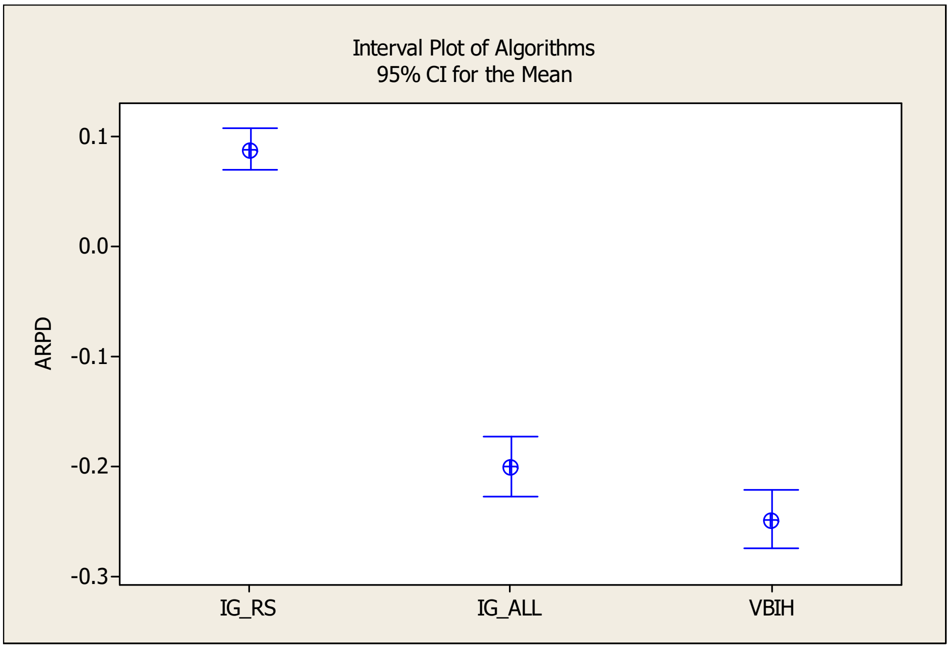

Figure 9. Interval plot at the 95% confidence level for large VRF instances.

In the Supplementary Materials, we summarize all the best-known solutions found for the first time by IG_{ALL} and VBIH algorithms. The VBIH algorithm further improves 230 out of 240 instances. In addition, 173 out of 240 instances are improved by the IG_{RS} algorithm, while the IG_{ALL} algorithm further improves 222 out of 240 instances. The IG_{ALL} algorithm improves six instances that are not improved by VBIH algorithm. Ultimately, 236 out of 240 instances are further improved by all algorithms within $45 \times n \times m$ time limits with the remaining four solutions being equal.

As mentioned before, IG_{ALL} algorithm is presented in [5], where they analyzed the performances of IG_{RS} and IG_{ALL} on both Taillard's [42] and large VRF instances. They observed that the results obtained by using Taillard's benchmark set, both algorithms do not present very significant differences with respect to the RPDs obtained. In fact, they have shown that both algorithms did not show any statistically significant differences. However, statistically significant differences between IG_{RS} and IG_{ALL} have been shown when large VRF instances are employed. In order to validate this observation, we have run three algorithms on Taillard's benchmark set with a stopping criterion $T_{max} = 45 \times n \times m$ milliseconds. Furthermore, we run three algorithms without the Taillard's speed up method and they are denoted as IG_{RS}*, IG_{ALL}* and VBIH*. The computational results are given in Table 9. As seen in Table 9, VBIH produced much better RPDs than IG_{RS} and IG_{ALL} algorithms when the Taillard's speed up method is employed since its overall RPD was 0.17 from the best-known solutions. However, IG_{RS} and IG_{ALL} algorithms do not show so many differences in terms of RPDs. Interval plots of the algorithms in Figure 10 show that differences in RFDs are not statistically significant because their confidence intervals do coincide. This suggests a fact that researches on PFSP and its variants should employ VRF benchmark suite to see differences in algorithms newly presented. Figure 10 also shows that the Taillard's speed up method is significantly effective for all three algorithms. During these runs, we were also able to find 3 new best-known solutions for the Taillard's benchmark suite (ta054 = 3719, ta55 = 3610, ta56 = 3680) and their permutations are also provided in the Supplementary Materials.

Table 9. Computational results of Taillard's instances with $T_{max} = 45 \times n \times m$ milliseconds (The bolds show better results).

	IG_{RS} Avg	IG_{RS} * Avg	IG_{ALL} Avg	IG_{ALL} * Avg	VBIH Avg	VBIH * Avg
20 × 5	0.00	0.00	0.00	0.00	0.00	0.00
20 × 10	0.01	0.00	0.00	0.00	0.00	0.01
20 × 20	0.01	0.01	0.00	0.00	0.00	0.01
50 × 5	0.00	0.00	0.00	0.00	0.00	0.00
50 × 10	0.34	0.43	0.40	0.43	0.26	0.31
50 × 20	0.57	0.79	0.53	0.71	0.33	0.53
100 × 5	0.00	0.00	0.00	0.00	0.00	0.00
100 × 10	0.10	0.19	0.04	0.11	0.02	0.09
100 × 20	0.82	1.33	0.89	1.23	0.54	0.94
200 × 10	0.05	0.14	0.03	0.05	0.03	0.05
200 × 20	1.04	1.46	0.82	1.29	0.55	1.02
500 × 20	0.47	0.92	0.35	0.75	0.26	0.64
Overall Avg.	0.28	0.44	0.26	0.38	**0.17**	**0.30**

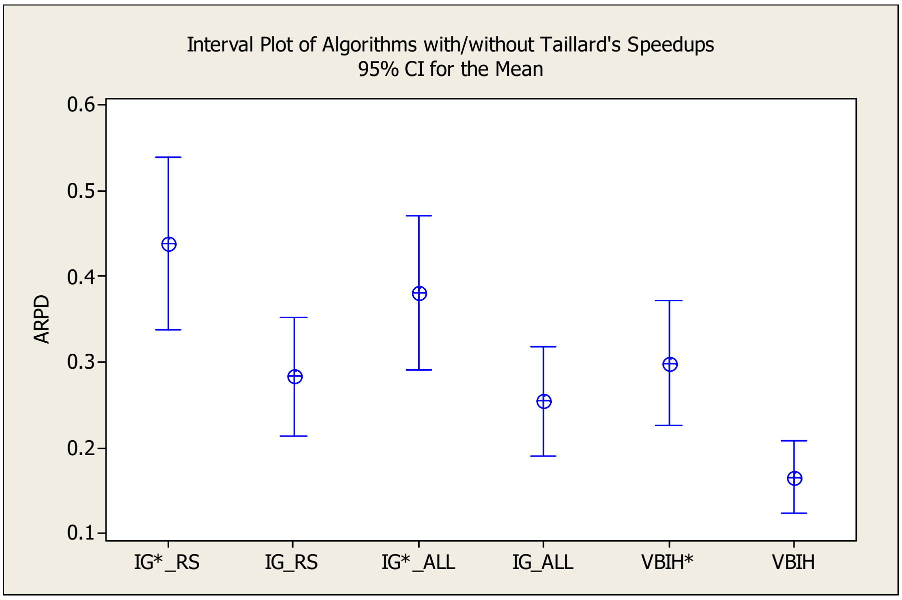

Figure 10. Interval plot at the 95% confidence level for Taillard's instances.

6. Conclusions

This paper presents a variable block insertion heuristic (VBIH) algorithm for solving the permutation flow shop scheduling problem (PFSP) with makespan criterion. In addition, we introduce mixed integer programming (MIP) and constraint programming (CP) models to solve the small benchmark set and to verify the results of our proposed heuristic algorithm. By employing the time limited CP model, we can find optimal solutions for some of small VRF instances for the first time in the literature. Furthermore, all algorithms can generate better solution values than upper those currently exist in the literature. We adapted a well-known speed-up method of Taillard and applied all the necessary parts while coding the heuristic algorithms. The parameters of the proposed VBIH algorithm is tuned through a design of experiments on randomly generated benchmark instances. Extensive computational results on two new VRF benchmark suites show that the VBIH algorithm is superior to the best performing algorithms from the literature.

CP model found and verify optimal solutions for 108 out of 240 small VRF instances, whereas 236 out of 240 large VRF benchmark instances are further improved by the VBIH and IG_{ALL} algorithms for the first time in this paper with remaining solutions being equal, which are also given in Appendix B

(Table A5). Furthermore, three instances of Taillard's benchmark suite are also further improved for the first time in this paper since 1993.

As future research, VBIH algorithm can be easily extended to other variants of the PFSPs such as no-idle, blocking and no-wait PFSP. In addition, other performance criteria can be considered such as total flow time and total tardiness. Furthermore, different meta-heuristic algorithms or matheuristics can be proposed to solve the PFSP.

Supplementary Materials: The following are available online at http://www.mdpi.com/1999-4893/12/5/100/s1.

Author Contributions: Conceptualization, M.F.T. and D.K.; methodology, Q.-K.P.; software, D.K., M.F.T. and Q.-K.P.; validation, D.K., and M.F.T. and L.G.; writing—original draft preparation, D.K.; writing—review and editing, L.G. and M.F.T.; supervision, M.F.T., Q.-K.P.; project administration, L.G; funding acquisition, L.G.

Funding: This research is partially supported by the National Natural Science Foundation of China (Grant No. 51435009).

Acknowledgments: M. Fatih Tasgetiren, Quan-Ke Pan and Liang Gao acknowledge the HUST Project in Wuhan in China. We also thank Eva Vallada for providing the permutations for large VRF benchmark suite so that we were able to check our code.

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

Appendix A

The processing times of Car8 instance is given in Table A1 in order to explain the speed-up method.

Table A1. Processing times of Car8 instance.

Jobs	Machines							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	456	654	852	145	632	425	214	654
2	789	123	369	678	581	396	123	789
3	654	123	632	965	475	325	456	654
4	321	456	581	421	32	147	789	123
5	456	789	472	365	536	852	654	123
6	789	654	586	824	325	12	321	456
7	654	321	320	758	863	452	456	789
8	789	147	120	639	21	863	789	654

We remove job 2 from the optimal solution and calculate the completion times of the partial solution, which is given in Table A2.

Table A2. Completion times of partial permutation.

$e_{j,k}$		Machines							
Job	**Position**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
7	1	654	975	1295	2053	2916	3368	3824	4613
3	2	1308	1431	2063	3028	3503	3828	4284	5267
8	3	2097	2244	2364	3667	3688	4691	5480	6134
5	4	2553	3342	3814	4179	4715	5567	6221	6344
1	5	3009	3996	4848	4993	5625	6050	6435	7089
6	6	3798	4650	5434	6258	6583	6595	6916	7545
4	7	4119	5106	6015	6679	6711	6858	7705	7828

After inserting job 2 to the 5th position $j = 5$, we calculate the completion times of heads below and they are summarized in Table A3:

$$f_{j,0} = 0$$
$$f_{j,k} = max\left\{f_{j,k-1}, e_{j-1,k}\right\} + p_{\pi_j,k}$$
$$f_{5,0} = 0$$
$$f'_{5,1} = max\left\{f_{5,0}, e_{4,1}\right\} + p_{5,1} = max\{0,\ 2553\} + 789 = 3342$$
$$f'_{5,2} = max\left\{f_{5,1}, e_{4,2}\right\} + p_{5,2} = max\{3342,\ 3342\} + 123 = 3465$$
$$f'_{5,3} = max\left\{f_{5,2}, e_{4,3}\right\} + p_{5,3} = max\{3465,\ 3814\} + 369 = 4183$$
$$f'_{5,4} = max\left\{f_{5,3}, e_{4,4}\right\} + p_{5,4} = max\{4183,\ 4179\} + 678 = 4861$$
$$f'_{5,5} = max\left\{f_{5,4}, e_{4,5}\right\} + p_{5,5} = max\{4861,\ 4715\} + 581 = 5442$$
$$f'_{5,6} = max\left\{f_{5,5}, e_{4,6}\right\} + p_{5,6} = max\{5442,\ 5567\} + 396 = 5936$$
$$f'_{5,7} = max\left\{f_{5,6}, e_{4,7}\right\} + p_{5,7} = max\{5936,\ 6221\} + 123 = 6344$$
$$f'_{5,8} = max\left\{f_{5,7}, e_{4,8}\right\} + p_{5,8} = max\{6344,\ 6344\} + 789 = 7133$$

Table A3. Completion times of heads for $\{7, 3, 8, 5, 2\}$ with $C_{max} = 7133$.

$f_{j,k}$		Machines							
Job	**Position**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
7	1	654	975	1295	2053	2916	3368	3824	4613
3	2	1308	1431	2063	3028	3503	3828	4284	5267
8	3	2097	2244	2364	3667	3688	4691	5480	6134
5	4	2553	3342	3814	4179	4715	5567	6221	6344
2	5	3342	3465	4183	4861	5442	5936	6344	7133

$$
\begin{gathered}
q_{j,m+1} = 0 \\
q_{j,k} = max\left\{q_{j,k+1}, q_{j+1,k}\right\} + p_{\pi_j,k} \\
q_{7,9} = 0 \\
q'_{7,8} = max\{q_{7,9}, q_{8,8}\} + p_{7,8} = max\{0,\ 0\} + 123 = 123 \\
q'_{7,7} = max\{q_{7,8}, q_{8,7}\} + p_{7,7} = max\{123,\ 0\} + 789 = 912 \\
q'_{7,6} = max\{q_{7,7}, q_{8,6}\} + p_{7,6} = max\{912,\ 0\} + 147 = 1059 \\
q'_{7,5} = max\{q_{7,6}, q_{8,5}\} + p_{7,5} = max\{1059,\ 0\} + 32 = 1091 \\
q'_{7,4} = max\left\{q_{7,5}, q_{8,4}\right\} + p_{7,4} = max\{1091,\ 0\} + 421 = 1512 \\
q'_{7,3} = max\left\{q_{7,4}, q_{8,3}\right\} + p_{7,3} = max\{1512,\ 0\} + 581 = 2093 \\
q'_{7,2} = max\{q_{7,3}, q_{8,2}\} + p_{7,2} = max\{2093,\ 0\} + 456 = 2549 \\
q'_{7,1} = max\left\{q_{7,2}, q_{8,1}\right\} + p_{7,1} = max\{2549,\ 0\} + 321 = 2870 \\
q_{6,9} = 0 \\
q'_{6,8} = max\{q_{6,9}, q_{7,8}\} + p_{6,8} = max\{0,\ 123\} + 456 = 579 \\
q'_{6,7} = max\{q_{6,8}, q_{7,7}\} + p_{6,7} = max\{579, 912\} + 321 = 1233 \\
q'_{6,6} = max\{q_{6,7}, q_{7,6}\} + p_{6,6} = max\{1233,\ 1059\} + 12 = 1245 \\
q'_{6,5} = max\{q_{6,6}, q_{7,5}\} + p_{6,5} = max\{1245,\ 1091\} + 325 = 1570 \\
q'_{6,4} = max\left\{q_{6,5}, q_{7,4}\right\} + p_{6,4} = max\{1570,\ 1512\} + 824 = 2394 \\
q'_{6,3} = max\left\{q_{6,4}, q_{7,3}\right\} + p_{6,3} = max\{2394,\ 2093\} + 586 = 2980 \\
q'_{6,2} = max\{q_{6,3}, q_{7,2}\} + p_{6,2} = max\{2980,\ 2549\} + 654 = 3634 \\
q'_{6,1} = max\left\{q_{6,2}, q_{7,1}\right\} + p_{6,1} = max\{3634,\ 2870\} + 789 = 4423 \\
q_{5,9} = 0 \\
q'_{5,8} = max\{q_{5,9}, q_{6,8}\} + p_{5,8} = max\{0,\ 579\} + 654 = 1233 \\
q'_{5,7} = max\{q_{5,8}, q_{6,7}\} + p_{5,7} = max\{1233, 1233\} + 214 = 1447 \\
q'_{5,6} = max\{q_{5,7}, q_{6,6}\} + p_{5,6} = max\{1447,\ 1245\} + 425 = 1872 \\
q'_{5,5} = max\{q_{5,6}, q_{6,5}\} + p_{5,5} = max\{1872,\ 1570\} + 632 = 2504 \\
q'_{5,4} = max\left\{q_{5,5}, q_{6,4}\right\} + p_{5,4} = max\{2504,\ 2394\} + 145 = 2649 \\
q'_{5,3} = max\left\{q_{5,4}, q_{6,3}\right\} + p_{5,3} = max\{2649,\ 2980\} + 852 = 3832 \\
q'_{5,2} = max\{q_{5,3}, q_{6,2}\} + p_{5,2} = max\{3832,\ 3634\} + 654 = 4486 \\
q'_{5,1} = max\left\{q_{5,2}, q_{6,1}\right\} + p_{5,1} = max\{4486,\ 4423\} + 456 = 4942
\end{gathered}
$$

Now, we calculate the completion times of tails as shown in Table A4.

Table A4. Completion times of tails for $\{2, 1, 6, 4\}$ with $C_{max} = 4942$.

$q_{j,k}$		Machines							
Job	**Position**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	6	4942	4486	3832	2649	2504	1872	1447	1233
6	7	4423	3634	2980	2394	1570	1245	1233	579
4	8	2870	2549	2093	1512	1091	1059	912	123

Now, we calculate $C_{max} = max_k(f_{j,k} + q_{j,k})$ at position j as follows:

$$
\begin{gathered}
C_{max} = max\{(3342 + 4942), (3465 + 4486), (4183 + 3832), (4861 + 2649), (5442 + 2504), (5936 + 1872), (6344 \\
+1447), (7133 + 1233)\} \\
C_{max} = max\{8284, 7951, 8015, 7979, 7832, 8108, 8366, 8366\} = 8366
\end{gathered}
$$

Appendix B

Table A5. New best solutions of our algorithms for Large VRF Instances (The bolds shows the new best known solutions).

Instance	Cmax	Best	Instance	Cmax	Best	Instance	Cmax	Best
100_20_1	6198	**6173**	300_60_1	20522	**20483**	600_40_1	33839	**33683**
100_20_2	6306	**6267**	300_60_2	20399	**20249**	600_40_2	33467	**33405**
100_20_3	6238	**6221**	300_60_3	20434	**20328**	600_40_3	33866	**33713**
100_20_4	6245	**6227**	300_60_4	20395	**20293**	600_40_4	33693	**33584**
100_20_5	6296	**6264**	300_60_5	20341	**20200**	600_40_5	33553	**33401**
100_20_6	6321	**6285**	300_60_6	20388	**20280**	600_40_6	33809	**33626**
100_20_7	6434	**6401**	300_60_7	20457	**20358**	600_40_7	33686	**33545**
100_20_8	6104	**6074**	300_60_8	20410	**20319**	600_40_8	33482	**33298**
100_20_9	6354	**6328**	300_60_9	20549	**20405**	600_40_9	33697	**33567**
100_20_10	6145	**6125**	300_60_10	20472	**20385**	600_40_10	33642	**33473**
100_40_1	7881	**7846**	400_20_1	21120	**21042**	600_60_1	36198	**35976**
100_40_2	8007	**7976**	400_20_2	21457	**21411**	600_60_2	36184	**35923**
100_40_3	7935	**7894**	400_20_3	21441	**21428**	600_60_3	36201	**35917**
100_40_4	7932	**7913**	400_20_4	21247	**21237**	600_60_4	36136	**36000**
100_40_5	8011	**7997**	400_20_5	21553	**21528**	600_60_5	36153	**36004**
100_40_6	8023	**7993**	400_20_6	21214	**21188**	600_60_6	36116	**35943**
100_40_7	8006	**7980**	400_20_7	21625	**21599**	600_60_7	36179	**35965**
100_40_8	7979	**7957**	400_20_8	21277	**21264**	600_60_8	36185	**35894**
100_40_9	7931	**7888**	400_20_9	21346	**21293**	600_60_9	36195	**35987**
100_40_10	7952	**7917**	400_20_10	21538	**21526**	600_60_10	36163	**35943**
100_60_1	9395	**9353**	400_40_1	23578	**23393**	700_20_1	36394	**36388**
100_60_2	9596	**9567**	400_40_2	23456	**23380**	700_20_2	36337	**36316**
100_60_3	9349	**9349**	400_40_3	23575	**23467**	700_20_3	36568	**36519**
100_60_4	9426	**9403**	400_40_4	23409	**23269**	700_20_4	36452	**36380**
100_60_5	9465	**9431**	400_40_5	23339	**23213**	700_20_5	36584	**36556**
100_60_6	9667	**9630**	400_40_6	23444	**23298**	700_20_6	36671	**36645**
100_60_7	9391	**9346**	400_40_7	23556	**23415**	700_20_7	36624	**36597**
100_60_8	9534	**9523**	400_40_8	23411	**23290**	700_20_8	36522	**36492**
100_60_9	9527	**9488**	400_40_9	23637	**23424**	700_20_9	36329	**36315**
100_60_10	9598	**9572**	400_40_10	23720	**23606**	700_20_10	36417	**36386**
200_20_1	11305	**11272**	400_60_1	25607	**25395**	700_40_1	38964	**38767**
200_20_2	11265	**11240**	400_60_2	25656	**25549**	700_40_2	38775	**38560**
200_20_3	11327	**11294**	400_60_3	25821	**25707**	700_40_3	38621	**38460**
200_20_4	11208	**11188**	400_60_4	25837	**25638**	700_40_4	38785	**38597**
200_20_5	11208	**11143**	400_60_5	25877	**25669**	700_40_5	38671	**38490**
200_20_6	11367	**11310**	400_60_6	25536	**25407**	700_40_6	38710	**38440**
200_20_7	11380	**11365**	400_60_7	25600	**25415**	700_40_7	38585	**38355**
200_20_8	11141	**11128**	400_60_8	25800	**25603**	700_40_8	39059	**38817**
200_20_9	11123	**11091**	400_60_9	25882	**25673**	700_40_9	38814	**38569**
200_20_10	11310	**11294**	400_60_10	25767	**25658**	700_40_10	38850	**38712**

Table A5. *Cont.*

Instance	Cmax	Best	Instance	Cmax	Best	Instance	Cmax	Best
200_40_1	13132	**13124**	500_20_1	26411	**26374**	700_60_1	41436	**41192**
200_40_2	13102	**13049**	500_20_2	26681	**26641**	700_60_2	41375	**41002**
200_40_3	13264	**13222**	500_20_3	26409	**26359**	700_60_3	41317	**41173**
200_40_4	13232	**13163**	500_20_4	26124	**26080**	700_60_4	41401	**41120**
200_40_5	13043	**12974**	500_20_5	26781	**26759**	700_60_5	41262	**41167**
200_40_6	13124	**13061**	500_20_6	26443	**26411**	700_60_6	41340	**41159**
200_40_7	13299	**13220**	500_20_7	26433	**26409**	700_60_7	40876	**40734**
200_40_8	13238	**13132**	500_20_8	26318	**26305**	700_60_8	41474	**41305**
200_40_9	13166	**13033**	500_20_9	26442	**26430**	700_60_9	41291	**41111**
200_40_10	13228	**13146**	500_20_10	26072	**26034**	700_60_10	41377	**41186**
200_60_1	14990	**14906**	500_40_1	28548	**28402**	800_20_1	41558	**41479**
200_60_2	14954	**14909**	500_40_2	28793	**28613**	800_20_2	41407	**41345**
200_60_3	15200	**15134**	500_40_3	28607	**28526**	800_20_3	41425	**41399**
200_60_4	15044	**14968**	500_40_4	28828	**28615**	800_20_4	**41426**	**41426**
200_60_5	15130	**15042**	500_40_5	28683	**28579**	800_20_5	41710	**41705**
200_60_6	15035	**14996**	500_40_6	28524	**28432**	800_20_6	42010	**41961**
200_60_7	15040	**15006**	500_40_7	28760	**28553**	800_20_7	41425	**41395**
200_60_8	14968	**14894**	500_40_8	28698	**28488**	800_20_8	41492	**41435**
200_60_9	15022	**14925**	500_40_9	28870	**28640**	800_20_9	41796	**41783**
200_60_10	15000	**14908**	500_40_10	28758	**28644**	800_20_10	41574	**41568**
300_20_1	16149	**16089**	500_60_1	30861	**30682**	800_40_1	43671	**43466**
300_20_2	16512	**16483**	500_60_2	30828	**30664**	800_40_2	43746	**43575**
300_20_3	16173	**16129**	500_60_3	31125	**30852**	800_40_3	43749	**43596**
300_20_4	16181	**16168**	500_60_4	30928	**30793**	800_40_4	43892	**43743**
300_20_5	16342	**16307**	500_60_5	30935	**30763**	800_40_5	43905	**43794**
300_20_6	16137	**16095**	500_60_6	31027	**30788**	800_40_6	43811	**43638**
300_20_7	16266	**16244**	500_60_7	30928	**30826**	800_40_7	43766	**43484**
300_20_8	16416	**16369**	500_60_8	30988	**30837**	800_40_8	43839	**43666**
300_20_9	16376	**16324**	500_60_9	30978	**30805**	800_40_9	43879	**43643**
300_20_10	16899	**16798**	500_60_10	31050	**30866**	800_40_10	43861	**43630**
300_40_1	18298	**18199**	600_20_1	31433	**31372**	800_60_1	46470	**46279**
300_40_2	18454	**18373**	600_20_2	31418	**31397**	800_60_2	46493	**46232**
300_40_3	18457	**18348**	600_20_3	**31429**	**31429**	800_60_3	46389	**46258**
300_40_4	18351	**18227**	600_20_4	31547	**31487**	800_60_4	46457	**46261**
300_40_5	18484	**18343**	600_20_5	31448	**31407**	800_60_5	46401	**46164**
300_40_6	18449	**18340**	600_20_6	31717	**31696**	800_60_6	46421	**46288**
300_40_7	18419	**18396**	600_20_7	**31527**	**31527**	800_60_7	46319	**46061**
300_40_8	18392	**18290**	600_20_8	31564	**31523**	800_60_8	46474	**46257**
300_40_9	18394	**18261**	600_20_9	31577	**31532**	800_60_9	46538	**46279**
300_40_10	18401	**18286**	600_20_10	31130	**31107**	800_60_10	46244	**46211**

References

1. Fernandez-Viagas, V.; Ruiz, R.; Framinan, J.M. A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *Eur. J. Oper. Res.* **2017**, *257*, 707–721. [CrossRef]
2. Pinedo, M.L. *Scheduling: Theory, Algorithms, and Systems*; Springer: New York, NY, USA, 2008.
3. Garey, M.R.; Johnson, D.S.; Sethi, R. The Complexity of Flowshop and Jobshop Scheduling. *Math. Oper. Res.* **1976**, *1*, 117–129. [CrossRef]
4. Ruiz, R.; Stützle, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *Eur. J. Oper. Res.* **2007**, *177*, 2033–2049. [CrossRef]
5. Dubois-Lacoste, J.; Pagnozzi, F.; Stützle, T. An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. *Comput. Oper. Res.* **2017**, *81*, 160–166. [CrossRef]
6. Ruiz, R.; Stützle, T. An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *Eur. J. Oper. Res.* **2008**, *187*, 1143–1159. [CrossRef]
7. Fernandez-Viagas, V.; Framinan, J. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Comput. Oper. Res.* **2014**, *45*, 60–67. [CrossRef]

8. Pan, Q.-K.; Tasgetiren, M.F.; Liang, Y.-C. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Comput. Ind. Eng.* **2008**, *55*, 795–816. [CrossRef]
9. Vallada, E.; Ruiz, R.; Framinan, J.M. New hard benchmark for flowshop scheduling problems minimising makespan. *Eur. J. Oper. Res.* **2015**, *240*, 666–677. [CrossRef]
10. Tasgetiren, M.F.; Pan, Q.-K.; Kizilay, D.; Velez-Gallego, M.C. A variable block insertion heuristic for permutation flowshops with makespan criterion. In Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), San Sebastian, Spain, 5–8 June 2017.
11. Shao, W.; Pi, D.; Shao, Z. Optimization of makespan for the distributed no-wait flow shop scheduling problem with iterated greedy algorithms. *Knowl. Based Syst.* **2017**, *137*, 163–181. [CrossRef]
12. Ding, J.-Y.; Song, S.; Gupta, J.; Zhang, R.; Chiong, R.; Wu, C. An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flowshop scheduling problem. *Appl. Soft Comput.* **2015**, *30*, 604–613. [CrossRef]
13. Li, X.; Yang, Z.; Ruiz, R.; Chen, T.; Sui, S. An iterated greedy heuristic for no-wait flow shops with sequence dependent setup times, learning and forgetting effects. *Inf. Sci.* **2018**, *453*, 408–425. [CrossRef]
14. Ribas, I.; Companys, R.; Tort-Martorell, X. An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega* **2011**, *39*, 293–301. [CrossRef]
15. Tasgetiren, M.F.; Kizilay, D.; Pan, Q.-K.; Suganthan, P.N. Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. *Comput. Oper. Res.* **2017**, *77*, 111–126. [CrossRef]
16. Fernandez-Viagas, V.; Leisten, R.; Framinan, J. A computational evaluation of constructive and improvement heuristics for the blocking flow shop to minimise total flowtime. *Expert Syst. Appl.* **2016**, *61*, 290–301. [CrossRef]
17. Tasgetiren, M.F.; Pan, Q.-K.; Kizilay, D.; Suer, G. A populated local search with differential evolution for blocking flowshop scheduling problem. In Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, Japan, 25–28 May 2015.
18. Ying, K.-C.; Lin, S.-W.; Cheng, C.-Y.; He, C.-D. Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems. *Comput. Ind. Eng.* **2017**, *110*, 413–423. [CrossRef]
19. Tasgetiren, M.F.; Pan, Q.-K.; Suganthan, P.N.; Buyukdagli, O. A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem. *Comput. Oper. Res.* **2013**, *40*, 1729–1743. [CrossRef]
20. Pan, Q.-K.; Ruiz, R. An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *Omega* **2014**, *44*, 41–50. [CrossRef]
21. Ding, J.-Y.; Song, S.; Wu, C. Carbon-efficient scheduling of flow shops by multi-objective optimization. *Eur. J. Oper. Res.* **2016**, *248*, 758–771. [CrossRef]
22. Öztop, H.; Tasgetiren, M.F.; Eliiyi, D.T.; Pan, Q.-K. Green Permutation Flowshop Scheduling: A Trade- off-Between Energy Consumption and Total Flow Time. In *Intelligent Computing Methodologies*; Springer: Cham, Switzerland, 2018; pp. 753–759.
23. Minella, G.; Ruiz, R.; Ciavotta, M. Restarted Iterated Pareto Greedy algorithm for multi-objective flowshop scheduling problems. *Comput. Oper. Res.* **2011**, *38*, 1521–1533. [CrossRef]
24. Ciavotta, M.; Minella, G.; Ruiz, R. Multi-objective sequence dependent setup times permutation flowshop: A new algorithm and a comprehensive study. *Eur. J. Oper. Res.* **2013**, *227*, 301–312. [CrossRef]
25. Pan, Q.-K.; Wang, L. Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. *Omega* **2012**, *40*, 218–229. [CrossRef]
26. Karabulut, K. A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops. *Comput. Ind. Eng.* **2016**, *98*, 300–307. [CrossRef]
27. Fernandez-Viagas, V.; Valente, J.M.S.; Framinan, J. Iterated-greedy-based algorithms with beam search initialization for the permutation flowshop to minimise total tardiness. *Expert Syst. Appl.* **2018**, *94*, 58–69. [CrossRef]
28. Pan, Q.-K.; Ruiz, R. Local search methods for the flowshop scheduling problem with flowtime minimization. *Eur. J. Oper. Res.* **2012**, *222*, 31–43. [CrossRef]
29. Tasgetiren, M.F.; Pan, Q.; Ozturkoglu, Y.; Chen, A.H.L. A memetic algorithm with a variable block insertion heuristic for single machine total weighted tardiness problem with sequence dependent setup times. In Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, 24–29 July 2016; pp. 2911–2918.

30. Subramanian, A.; Battarra, M.; Potts, C.N. An Iterated Local Search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. *Int. J. Prod. Res.* **2014**, *52*, 2729–2742. [CrossRef]
31. Xu, H.; Lü, Z.; Cheng, T.C.E. Iterated Local Search for single-machine scheduling with sequence-dependent setup times to minimize total weighted tardiness. *J. Sched.* **2014**, *17*, 271–287. [CrossRef]
32. Fernández, M.Á.G.; Palacios, J.; Vela, C.; Hernández-Arauzo, A. Scatter search for minimizing weighted tardiness in a single machine scheduling with setups. *J. Heuristics* **2017**, *23*, 81–110.
33. Tasgetiren, M.F.; Pan, Q.-K.; Kizilay, D.; Gao, K. A Variable Block Insertion Heuristic for the Blocking Flowshop Scheduling Problem with Total Flowtime Criterion. *Algorithms* **2016**, *9*, 71. [CrossRef]
34. Manne, A.S. On the Job-Shop Scheduling Problem. *Oper. Res.* **1960**, *8*, 219–223. [CrossRef]
35. Taillard, E. Some efficient heuristic methods for the flow shop sequencing problem. *Eur. J. Oper. Res.* **1990**, *47*, 65–74. [CrossRef]
36. Johnson, S.M. Optimal Two and Three Stage Production Schedules with Set-Up Time Included. *Nav. Res. Logist. Q.* **1954**, *1*, 61–68. [CrossRef]
37. Nawaz, M.; Enscore, E.E.; Ham, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **1983**, *11*, 91–95. [CrossRef]
38. Osman, I.; Potts, C.N. Simulated Annealing for Permutation Flow-Shop Scheduling. *Omega* **1989**, *17*, 551–557. [CrossRef]
39. Rad, S.F.; Ruiz, R.; Boroojerdian, N. New high performing heuristics for minimizing makespan in permutation flowshops. *Omega* **2009**, *37*, 331–345. [CrossRef]
40. Tasgetiren, M.F.; Pan, Q.-K.; Suganthan, P.N.; Chua, T.J. A differential evolution algorithm for the no-idle flowshop scheduling problem with total tardiness criterion. *Int. J. Prod. Res.* **2011**, *49*, 5033–5050. [CrossRef]
41. Montgomery, D.C. *Design and Analysis of Experiments*, 2nd ed.; Wiley: New York, NY, USA, 1984.
42. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [CrossRef]