algorithms

MDPI

Article

A Variable Block Insertion Heuristic for the Blocking Flowshop Scheduling Problem with Total Flowtime Criterion

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

1 International Logistics Management Department, Yasar University, Izmir 35100, Turkey
2 Department of Industrial and Manufacturing System Engineering, School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China; panquanke@hust.edu.cn
3 Industrial Engineering Department, Yasar University, Izmir 35100, Turkey; damla.kizilay@yasar.edu.tr
4 School of Electrical and Electronics, Nanyang Technological University, Singapore 639798, Singapore; kgao001@e.ntu.edu.sg
* Correspondence: fatih.tasgetiren@yasar.edu.tr; Tel.: +90-530-827-3715

Academic Editor: Javier Del Ser Lorente
Received: 30 June 2016; Accepted: 8 October 2016; Published: 20 October 2016

Abstract: In this paper, we present a variable block insertion heuristic (VBIH) algorithm to solve the blocking flowshop scheduling problem with the total flowtime criterion. In the VBIH algorithm, we define a minimum and a maximum block size. After constructing the initial sequence, the VBIH algorithm starts with a minimum block size being equal to one. It removes the block from the current sequence and inserts it into the partial sequence sequentially with a predetermined move size. The sequence, which is obtained after several block moves, goes under a variable local search (VLS), which is based on traditional insertion and swap neighborhood structures. If the new sequence obtained after the VLS local search is better than the current sequence, it replaces the current sequence. As long as it improves, it keeps the same block size. However, if it does not improve, the block size is incremented by one and a simulated annealing-type of acceptance criterion is used to accept the current sequence. This process is repeated until the block size reaches at the maximum block size. Furthermore, we present a novel constructive heuristic, which is based on the profile fitting heuristic from the literature. The proposed constructive heuristic is able to further improve the best known solutions for some larger instances in a few seconds. Parameters of the constructive heuristic and the VBIH algorithm are determined through a design of experiment approach. Extensive computational results on the Taillard's well-known benchmark suite show that the proposed VBIH algorithm outperforms the discrete artificial bee colony algorithm, which is one of the most efficient algorithms recently in the literature. Ultimately, 52 out of the 150 best known solutions are further improved with substantial margins.

Keywords: meta-heuristics; blocking flowshop; block insertion heuristic; variable local search; constructive heuristics

1. Introduction

There have been extensive studies about permutation flowshop scheduling (PFSP) in the literature with many important applications in manufacturing and service systems [1–4]. The traditional PFSP is concerned with scheduling n jobs through m machines in such a way that the same sequence is applied to each machine. An important assumption is such that work-in-process inventory is allowed since there are infinite buffer capacities amongst consecutive machines. Hence, allowing jobs to be waiting in front of machines for their next operations words. On the other hand, if any storage capacity amongst machines is not available, the traditional PFSP is said to be blocking flowshop scheduling problem

(BFSF) [5]. In this case, a job cannot leave the machine unless the next machine is free. Its industrial applications can be found in [5,6]. A comprehensive review on flowshop scheduling with blocking and no-wait constraint can be found in Hall and Siriskandarajah [7].

The makespan criterion for the BFSP is commonly used in the literature. This problem is also denoted as $F_m/blocking/C_{max}$ with the notation of Graham et al. [8] in the literature. Although Gilmore–Gomory's algorithm [9] can solve it optimally with two machine case ($m = 2$), it is proven to be NP-Hard by Hall and Sriskandarajah [7] when $m > 2$. For this reason, efforts have been devoted on developing heuristic and meta-heuristic approaches for scheduling a large number of jobs, which is commonly needed for real-life problems.

Since the problem is NP-Hard, constructive heuristics and meta-heuristic algorithms have been attracted attention to solve the BFSP with the makespan criterion. Regarding the constructive heuristics, McCormick et al. [10] presented a profile fitting (PF) heuristic to solve BFSP with the performance measure of minimization of cycle time. The PF heuristic greedily establishes a sequence with the next job having minimum idle and blocking times on machines. Leisten [11] presented a similar heuristic for flowshop scheduling problems with finite and unlimited buffers in order to maximize the buffer usages and to minimize the machine blocking times. However, it was not able to obtain better results than the NEH heuristic, which is initially proposed in [12] to solve the traditional PFSP. Depending on the makespan properties defined by Ronconi and Armentano [13], Ronconi [6] presented a note on constructive heuristics and proposed three constructive heuristics, called MM, MM combined with NEH (MME), and PF combined with NEH (PFE), respectively for the BFSP with the makespan criterion. It was shown that the MME and PFE heuristics generated better results than the NEH heuristics up to 500 jobs and 20 machines. Abadi et al. [14] proposed an improvement heuristic to minimize cycle time and Ronconi and Henriques [15] considered the minimization of total tardiness in a flowshop with blocking and presented some constructive heuristics with promising results. Furthermore, Pan and Wang [16] developed some effective heuristics based on PF approach. These PF-based heuristics are inspired from LR heuristic proposed by Liu and Revees [17] for the PFSP with the total flowtime criterion. In their work, the PF heuristic is combined with the partial NEH implementation and they called the heuristics as PF_NEH(x), WPF_NEH(x) and PW_NEH(x), where x is the number of sequences generated by considering the first x number of jobs in the initial order of jobs. In order not to ruin good characteristics of the PF heuristic, the NEH heuristic is applied to only the last δ jobs. Their PW_NEH(x) heuristic with $x = 5$ was substantially better than NEH, MME, PFE, WPFE and PWE heuristics from the literature. Regarding the meta-heuristic algorithms for BFSP with the makespan criterion, the literature can be found in [18–33].

The makespan criterion is a firm-oriented performance measure, which aims at minimizing the idle times on the machines, thus resulting in a maximization of machine utilization. However, total flowtime and total tardiness criteria are both customer-oriented performance measures, which aim at minimizing the waiting times of jobs amongst machines in order to finish the jobs as early as possible, thus resulting in a maximization of the customer satisfaction. Since the tardiness is measured by the difference between due date and flowtime, both performance measures are equivalent.

For the total tardiness criterion, a few papers can be found in [34,35]. Regarding the total flowtime criterion, a few papers can be found in the literature. A hybrid harmony search is presented in [36]. A discrete artificial bee colony algorithm is presented in [37]. An iterated greedy algorithm is developed in [38], and a branch and bound is presented in [39] for solving small size of instances. Very recently, a GRASP and a discrete artificial bee colony (DABC_RCT) algorithm are developed in [40,41], which outperformed the existing algorithms from the literature.

In this paper, we present a variable block insertion heuristic (VBIH) algorithm to solve the BFSP with the total flowtime criterion. Through extensive computational analyses on Taillard's well-known benchmark suite, we demonstrate that the proposed VBIH algorithm outperforms the recent best performing DABC_RCT algorithm from the literature. Ultimately, 52 out of 150 problem instances are further improved with substantial margins.

The rest of the paper is organized as follows. In Section 2, the blocking flowshop scheduling problem with speed-up method is formulated. Section 3 presents the components of the VBIH algorithm. Section 4 presents the design of experiment approach for parameter tuning. The computational results and comparisons are provided in Section 5. Finally, Section 6 gives the concluding remarks.

2. Blocking Flow Shop Scheduling Problem

In the blocking flowshop scheduling problem, n jobs from the set $J = \{1, 2, .., n\}$ have to be processed on m machines from the set $M = \{1, 2, .., m\}$ with the same permutation on each machine without any intermediate buffer. Each job j has a processing time on machine k, which is denoted as $p_{j,k}$. The setup time is assumed to be included in the processing time. Only a single job can be processed on each machine. Since waiting times are not allowed in the flowshop due to no intermediate buffers, jobs cannot leave machines after completing their operations until next machines are free. In other words, if the next machine is busy, then the current job on the machine must be blocked since there is no intermediate buffer amongst machines. The goal is to obtain a permutation, which will be applied to each machine and the total flowtime (TFT) is to be minimized. Given a job sequence $\pi = \{\pi_1, \pi_2, .., \pi_n\}$, the departure time $d_{\pi_j,k}$ of job π_j on machine k can be computed by following Ronconi [6] as follows:

$$d_{\pi_1,0} = 0 \tag{1}$$

$$d_{\pi_1,k} = d_{\pi_1,k-1} + p_{\pi_1,k} \; k = 1, .., m-1 \tag{2}$$

$$d_{\pi_j,0} = d_{\pi_j-1,1} \; j = 2, .., n \tag{3}$$

$$d_{\pi_j,k} = max\left\{d_{\pi_j,k-1} + p_{\pi_j,k}, d_{\pi_j-1,k+1}\right\} \; j = 2, .., n \; and \; k = 1, .., m-1 \tag{4}$$

$$d_{\pi_j,m} = d_{\pi_j,m-1} + p_{\pi_j,m} \; j = 1, .., n \tag{5}$$

where $d_{\pi_j,0}$, $j = 1, .., n$ denotes the starting time of job j on the first machine. Since $C_{\pi_j,m} = d_{\pi_j,m}$ for all j, then the total flowtime of the sequence π can be given as $TFT(\pi) = \sum_{j=1}^{n} C_{\pi_j,m}$. Briefly, the objective is to determine a sequence π^* in the set of all sequences Π such that $TFT(\pi^*) \leq TFT(\pi) \; \forall \pi \in \Pi$.

There are two neighborhood structures in scheduling problems in general. These are based on insertion and swap neighborhood structures. As known, the computational complexity of both neighborhood structures is $O(n^3 m)$. For the traditional PFSP with the total flowtime criterion, Lia et al. [42] proposed a speed-up method for the swap and insertion neighborhood structures. They showed that the proposed speed-up method makes a reduction in the CPU times up to 40 to 50 percent. Based on their idea, we develop a fast fitness function calculation for insertion and swap moves. Suppose that we have sequence π. We first calculate the departure times of each job on each machine and store them in a matrix denoted as $D_{\pi_j,k}$. Given the current sequence as $\pi = \{1, 2, 3, 4, 5\}$. Now, suppose that we would like to interchange job 3 at position 3 with job 4 at position 4. Due to the fact that the departure times are stored in $D_{\pi_j,k}$ in advance until job 3 at position 3, we do not need to re-calculate departure times until job 3 at position 3, again. In other words, after interchanging those two jobs, the departure times of the new sequence $\pi = \{1, 2, 4, 3, 5\}$ can be calculated as follows. First, we copy the departure times from $D_{\pi_j,k}$ matrix with $d_{\pi_j,k} = D_{\pi_j,k} \; for \; j = 1, .., 3 \; and \; k = 1, .., m$. Then, we calculate the departure times starting from position 3 to 5 with $d_{\pi_j,k}$, $j = 3, .., 5$, $k = 1, .., m$.

In order to generalize it, we first determine two randomly chosen positons $pos1$ and $pos2$ such that $(pos1 < pos2) \in (1, n)$. Our swap function, $\pi = Swap\left(\pi^0, pos1, pos2\right)$, interchanges jobs $\pi^0_{j=pos1}$ and $\pi^0_{j=pos2}$; then, store the new sequence on a temporary sequence π. Finally, our fast fitness function, $f_{fast}\left(\pi, pos1\right)$, copies the departure times from $D_{\pi_j,k} \; matrix$ until position $j = pos1$ and then, re-calculates the departure times starting from position $pos1$ to the end of the remaining part of the sequence. After swapping jobs and storing them in a sequence π, the fast fitness function is given in Figure 1.

The same fast fitness function can be used for insertion moves, too. Before we get into details, we need to clarify that our insertion move function utilizes forward or backward insertion move with an equal probability of 0.5. We again determine two random positions $pos1$ and $pos2$ such that $(pos1 < pos2) \in (1, n)$. Our insertion function, $\pi = Insert\left(\pi^0, pos1, pos2\right)$, either removes job $\pi^1_{j=pos1}$ at position $pos1$ and inserts it into $pos2$ in a temporary sequence π (indicating a forward insertion move) or removes job $\pi^1_{j=pos2}$ at position $pos2$ and inserts it into pos1 in π (indicating a backward insertion move). In both cases, one does not need to re-calculate the departure times until position $pos1$, which are copied from the $D_{\pi_j,k}$ matrix. In other words, $f_{fast}\,(\pi, pos1)$ function in Figure 1 can also be used for insertion moves.

Procedure FastFitness Calculation

Step1. $if\ (pos1 = 1)\ then\ compute\ fitness\ of\ whole\ permutation$
$else\ go\ to\ Step2$

Step2. $Copy\ departure\ times\ from\ D_{\pi_j,k}\ until\ pos1$
$d_{\pi_j,k} = D_{\pi_j,k}\ for\ j = 1,..,pos1\ and\ k = 1,..,m.$

Step3. $Compute\ departure\ times\ from\ pos1\ until\ n$
$d_{\pi_j,k} = max\left\{d_{\pi_j,k-1} + p_{\pi_j,k}, d_{\pi_{j-1},k+1}\right\}\ for\ j = pos1,..,n\ and\ k = 1,..,m-1.$

Step4. $Compute\ departure\ times\ on\ the\ last\ machine\ m$
$d_{\pi_j,m} = d_{\pi_j,m-1} + p_{\pi_j,m}\ for\ j = 1,..,n$

Step5. $Compute\ total\ flowtime$
$TFT = 0;\ \mathrm{TFT} = \mathrm{TFT} + d_{\pi_j,m}\ for\ j = 1,..,n$

Step6. $Return\ TFT\ and\ \pi$

Figure 1. $f_{fast}\,(\pi, pos1)$ function.

3. Variable Block Insertion Heuristic

Traditional local search algorithms are based on swap and insertion neighborhood structures. The swap operator exchanges two jobs in a sequence, whereas the insertion operator removes a job from a sequence and inserts it into another position in the sequence. Recently, local search algorithms, which are based on block moves, are presented for the single machine scheduling problem in the literature. A block move of 2-edge exchange was presented in Kirlik and Oguz [43]. Then, Subramanian et al. [44] developed five different neighborhoods: swap, insertion, edge-insertion, 3-block insertion and 3-block reverse. The basic idea is to use larger neighborhood move operators rather than the swap and insertion move operators. Relying on this idea, Xu et al. [45] presented a neighborhood structure called a *Block Move* in which l consecutive jobs (called a block) are inserted into another position in the sequence. They represent a block move by a $triplet\ \ (i,\ k,\ l)$, where i represents the position of the first job of the block, k denotes the target position of the block to be inserted and l represents the size of the block. Note that one edge insertion, two edge-insertion and 3-block insertion corresponds to the block move neighborhoods with $l = 1$, $l = 2$, and $l = 3$. Similarly, Gonzales and Vela [46] developed a variable neighborhood descent algorithm by using three block move neighborhoods were developed and used in a memetic algorithm.

Inspiring from the algorithms above, we propose a similar, but a different block insertion heuristic, which we call it a variable block insertion heuristic (VBIH) algorithm. We denote two block sizes as bS_{min} and bS_{max}, where bS_{min} is the minimum block size and bS_{max} is the maximum block size. In addition, we define a block move insertion size with a maximum insertion move size denoted as mS_{max}. The VBIH algorithm removes a block of jobs with size, bS, from the current sequence starting from $bS_{min} = 1$, then it makes a number mS_{max} of block insertion moves randomly in the partial sequence, which we denote it as the block insertion move procedure, $(bIM())$. It chooses the best one amongst a number mS_{max} of block insertion moves. The sequence obtained after $bIM()$ procedure goes under a variable local search (VLS), which is based on traditional insertion and swap neighborhood structures. If the new sequence obtained after the VLS local search is better than the current sequence, it replaces the current sequence. As long as it improves, it keeps the same block size, i.e., $bS = bS$. Otherwise, the block size is incremented by one, i.e., $bS = bS + 1$ and a simulated annealing type of acceptance criterion is used to accept the inferior solution in order to escape from local minima. This process is repeated until the block size reaches at the maximum block size, i.e., $bS \leq bS_{max}$. The outline of the VBIH algorithm is given in Figure 2. Note that π^R is the reference sequence; T is temperature parameter for the acceptance criterion, and tPF_ NEH(x) is a constructive heuristic providing an initial sequence to the VBIH algorithm. They will be explained in detail in the subsequent sections.

```
Procedure VBIH
π = tPF_NEH(x)
π_best = π^R = π
while (NotTermination) do
    bS = bS_min = 1
    do{
        π_1 = bIM(π, bS, mS_max)
        π_2 = VLS(π_1, f(π_1))
        if (f(π_2) < f(π)) then do{
            π = π_2
            bS = bS
            if f(π_2) < f(π_best) then do{
                π_best = π^R = π_2
            }endif
        else{
            bS = bS + 1
            if (r < exp{−(f(π) − f(π_2))/T})
                π = π_2
        }endif
    }while(bS ≤ bS_max)
}endwhile
return π_best and f(π_best)
endprocedure
```

Figure 2. Variable block insertion heuristic.

3.1. Initial Solution

The initial solution for the VBIH algorithm is generated by the tPF_ NEH(x) heuristic, which is proposed for the first time in this paper in the literature. It is a novel modification of the PF_ NEH(x) heuristic presented in Pan and Wang [16]. In the PF_ NEH(x) heuristic, the cost function to determine the next job to be scheduled is comprised of the total idle and blocking times. Note that the cost function devised for any constructive heuristic has a significant impact on the results. For this reason, Tasgetiren et al. [47] developed a PFT_ NEH(x) heuristic for the BFSP with the makespan criterion, which was inspired from the PF_ NEH(x) heuristic in [16]. In [47], a new cost function was devised by adding the departure time of the last job on the last machine to the total idle and blocking times. In this paper, we extend the PFT_ NEH(x) heuristic to the total flowtime criterion and denote it as the tPF_ NEH(x) heuristic. As in the PFT_ NEH(x) heuristic, we consider the total idle and blocking times as a part of the cost function. In addition, we also consider the sum of the departure times on all machines of the last job that could be inserted into the partial sequence, which is one of the main contribution of this paper.

To implement the tPF_ NEH(x) heuristic, an initial order of jobs should be determined. To construct the initial order, we use the front delay and total processing times of each job as in Ribas et al. [41]. The following measure is employed to construct the initial order of jobs of jobs:

$$iO(j) = \frac{2}{m-1}\left(\sum_{k=1}^{m}(m-k)p_{\pi_j,k}\right) + \sum_{k=1}^{m} p_{\pi_j,k} \quad (6)$$

In order to establish the initial order of jobs, we sort the $iO\,(j)$ values with an ascending order. Now, the number x of new sequences can be generated from the initial order of jobs as follows. Since the first job has an impact on the solution quality, the first job of the initial order is taken as the first job of the new sequence and the tPF_ NEH(x) heuristic is applied to generate the new solution. Then, the second job of the initial order is taken as the first job of the new sequence and tPF_ NEH(x) heuristic is applied to generate another new solution. This is repeated x times and the number x of new sequences will be generated. Of course, the best one amongst them is chosen as the initial solution for the VBIH algorithm.

The proposed constructive heuristic can be summarized as follows. Suppose that π_{i-1} jobs have already been scheduled and a partial sequence, $\pi = \{\pi_1, \pi_{2,..}, \pi_{i-1}\}$, is obtained. Clearly, job π_i will be the next job to be inserted into the partial sequence π_{i-1}. It can be any job from all jobs in the set U of the unscheduled jobs. To select the job π_i, we need a cost measure (CM). For the BFSP with the total flowtime criterion, we propose a new cost function consisting of the total idle and blocking times as well as the sum of the departure times of job π_i on all machines. We first calculate the total idle and blocking time as follows:

$$IT_i = \sum_{k=1}^{m} \left(d_{i,k} - d_{i-1,k} - p_{\pi_i,k}\right) \tag{7}$$

Then, we calculate the indexed sum of the departure times of job π_i on all machines as follows:

$$SD_i = \sum_{k=1}^{m} \frac{md_{i,k}}{k + i\,(m-k)\,/\,(n-2)} \tag{8}$$

Now, we define our cost function as follows:

$$CM_i = (1-\mu) \times IT_i + \mu \times SD_i \tag{9}$$

Note that we also employ an index function, $m/\,(k + i \times (m-k)\,/\,(n-2))$, to give a weight to departure time on each machine. Then, the job with the smallest sum of CM_i amongst all jobs in U is determined as the ith job to be inserted to the partial sequence π_{i-1}. Figure 3 outlines the procedure of the proposed heuristic. As seen in Figure 3, the $tPF\ (\pi^*, h)$ procedure takes the initial sequence as π^* and the job-index as y sent by the tPF_ NEH(x) heuristic. By using the job-index y, the x number of sequences will be generated.

Procedure $tPF(\pi^*, y)$

*Step*1. *Do for* $(j = 1 \text{ to } n)\ bS(j) = false$
*Step*2. *Set* $nJob = \pi^*_y, \pi_1 = nJob,\ U = J - \{\pi_1\}$ *and* $bS[\pi_1] = true$
*Step*2. *Calcualte the departure times* $d_{1,k}$ *of job* π_1 *for* $k = 1,..,m$
for $(i = 2 \text{ to } n)$ *do*
    $tempF = INT\_MAX$
    for $(j = 1 \text{ to } n)$ *do*
      //Consider only jobs in U with bS(j) flags are false
      - *if* $(bS(j) = true)$ *then continue*
      - $\pi_i = j$
      - *Calculate the departure times* $d_{i,k}$ *for* $k = 1,2,..m$
      - *Calculate the sum of idle and blocking times*
        $IT_i = \sum_{k=1}^{m}\left(d_{i,k} - d_{i-1,k} - p_{\pi_i,k}\right)$
      - *Calculate the indexed sum of departure times on all machines*
      - $SD_i = \sum_{k=1}^{m} \frac{md_{i,k}}{k+i(m-k)/(n-2)}$
      - *Now, calculate the cost function*
        $CM_i = (1-\mu) \times IT_i + \mu \times SD_i$
      - *Determine the best job in U*
        if $(CM_i < tempF)$
          $tempF = CM_i$
          $nJob = \pi_i$
        endif
    Endfor
  $\pi_i = nJob$
  $f(\pi_i) = tempF$
  //Remove job π_i ***from U***
  $U = J - \{\pi_i\}$
  $bS[\pi_i] = true$
EndFor
Return π *and* $f(\pi)$

Figure 3. tPF heuristic.

Pan and Wang [16] showed that the PF heuristic is very effective, but applying the NEH heuristic to the whole sequence can worsen the objective function, i.e., total flowtime. To avoid it, they applied the NEH heuristic to only the last δ jobs. As can be seen above, the tPF_ NEH(x) heuristic has two parameters, namely, δ and μ. Since the number x of sequences that will be generated are not so costly until the number of jobs is less than 200, i.e., $n \leq 200$, in terms of CPU times, we determined it as $if\ \ (n \leq 200)\ \ then\ x = n$; ***else*** $x = 20$. Another distinction between PF_ NEH(x) and tPF_ NEH(x) is about how to choose the best one amongst x number of sequences. Since even the partial NEH implementation may result in an inferior solution when compared to tPF heuristic, we check both solutions and keep the better one. Note that in all figures throughout the paper, $f\ (\pi)$ and r corresponds to the total flowtime (TFT) value of a sequence π and a uniform random number between 0 and 1, respectively. The tPF_ NEH(x) heuristic is outlined in Figure 4.

$Procedure\ tPF\_NEH(x)$

Step1. $Sort\ the\ jobs\ with\ iO(j)\ values\ with\ an\ ascending\ order$
Step2. $Denote\ initial\ solution\ as\ \pi^1$
Step3 $if\ (n \leq 200)\ then\ x = n$ else $x = 20$.
Step4. $For\ (h = 1\ to\ x)\ do$

- $Take\ \pi_h\ as\ the\ first\ job$
- $Generate\ a$ sequence $by\ tPF\ heuristic\ \pi^2 = tPF(\pi^1, h)$
- $For\ (i = n - \delta + 1\ to\ n\ )do$
    - $Remove\ job\ \pi_i\ from\ sequence\ \pi^2$
    - $Evaluate\ all\ possible\ position\ by\ inserting\ \pi_i\ in\ \pi^2$
    - $Insert\ \pi_i\ in\ \pi^2\ with\ the\ lowest\ total\ flowtime\ (TFT)$
- $EndFor$
- $Obtain\ the\ sequence\ \pi^3$
- $if\ \left(f(\pi^2) < f(\pi^3)\right)\ \pi^h = \pi^2; else\ \pi^h = \pi^3$
- $Record\ the\ sequence\ as\ \pi^h$
- $Return\ the\ best\ one\ with\ the\ lowest$ TFT $amongst\ \{\pi^1, \pi^2, .., \pi^x, \}$

Figure 4. tPF_NEH(x) heuristic.

3.2. Block Insertion Move Procedure

$bIM\ (\pi,\ bS,\ mS_{max})$ procedure is a core function in the VBIH algorithm. It takes the sequence π, the block size, bS and the maximum insertion move size, mS_{max} as parameters. The procedure randomly removes a block of jobs with size, bS from the current sequence π, which is denoted as π^b. Then, the partial sequence after removal will be denoted as $\pi^p = J - \left\{\pi^b\right\}$. Then, the procedure carries out a number mS_{max} of block insertion moves with the block size, bS. In other words, the block π^b is inserted in the partial sequence π^p randomly in which the maximum size of the insertion moves is mS_{max}. Finally, it chooses the best one amongst a number mS_{max} of block insertion moves. The $bIM\ (\pi,\ bS, mS_{max})$ procedure is given in Figure 5.

In order to ease the understanding of the block insertion move procedure, we give the following example. Suppose that we have a current sequence $\pi = \{7, 4, 1, 8, 2, 6, 5, 3\}$. In addition, suppose that the block size is $bS = 3$ and maximum insertion move size is $mS_{max} = 3$. Suppose that we randomly choose a block $\pi^b = \{8, 2, 6\}$. Then, the partial sequence will be $\pi^p = \{7, 4, 1, 5, 3\}$. Without considering the total flowtime of the sequence π, we randomly choose three positions, for instance, 1, 2, 5, and insert the block in these positions. Hence we have three sequences as $\pi_1 = \{8, 2, 6, 7, 4, 1, 5, 3\}$, $\pi_2 = \{7, 8, 2, 6, 4, 1, 5, 3\}$, and $\pi_3 = \{7, 4, 1, 5, 8, 2, 6, 3\}$. Amonst these three sequences we take the one with the lowest total flowtime criterion. Note that the same speed up methods explained before can be used to accelerate the insertion procedure. For example, after removing the block $\pi^b = \{8, 2, 6\}$, $D_{j,k}$ matrix of partial sequence π^p is once calculated, and then the fast fitness calculation procedure is used to accelerate the insertion procedure.

```
Procedure bIM(π, bS, mS_max)
tempF = INT_MAX
π_1 = π
π^b = remove the block of jobs with size bS from π_1
π^p = partial solution after removal
for (i = 1 to mS_max) do
    pt = Choose a position in π^p randomly
    π_2 = Insert block π^b in position pt of π^p
    if (f(π_2) < tempF) then do
        π_1 = π_2
        tempF = f(π_1)
    endif
endfor
f(π_1) = tempF
return π_1 and f(π_1)
endprocedure
```

Figure 5. Block insertion procedure.

3.3. Variable Local Search

The traditional variable neighborhood search (VNS) algorithm was developed in [48] and successfully applied to scheduling problems in [49–53]. Very recently, a variable local search (VLS) algorithm is developed by Ribas et al. [32,40,41]. The VLS algorithm is a novel and different from the traditional VNS algorithms in such a way that when a sequence is improved by any randomly chosen neighborhood structure, it switches to another neighborhood structure. For example, if a sequence is improved by the LS1 local search (i.e., swap neighborhood), it systematically switches to LS2 local search (i.e., insertion neighborhood), which is the difference between traditional VNS and VLS algorithms. As seen in Figure 6, if a sequence is improved by the local search type "LS", it switches to $LS = 1 - LS$. Suppose that the LS is randomly chosen as 1, then if the sequence is improved by the insertion neighborhood, the variable local search type is switched to 0 by $LS = 1 - 1 = 0$. This is different from the traditional VNS algorithms, where if a neighborhood improves, that neighborhood is kept for the search process and it switches to the second neighborhood if it fails, which is a common sense to follow. Note that, at least one time, both neighborhoods are ensured to be applied by using a *Counter*. The VLS local search is given in Figure 6.

```
Procedure VLS(π, f(π))
    Counter = 0
    if (r < 0.5) then do
        LS = 0
    else
        LS = 1
    endif
    //Apply variable local search as long as it improves
    do{
        Counter = Counter + 1
        C^0_TFT = f(π)
        if (LS = 0) then do
            π_1 = LS1(π)
        else
            π_1 = LS2(π)
        endif
        if (f(π_1) < C^0_TFT or Counter = 1) then do
            LS = 1 − LS
            π = π_1
            C^0_TFT = f(π_1)
        endif
    }while(true)
return π and f(π)
```

Figure 6. Variable local search (VLS) algorithm.

Note that the VLS algorithm employs two powerful local search algorithms in the VBIH algorithm, namely, referenced insertion scheme (RIS) and referenced swap scheme (RSS). Briefly, in the VLS algorithm, the RIS local search is used as LS1 whereas the RSS local search is employed as LS2. The RIS and RSS local search algorithms are outlined in Figures 7 and 8, respectively.

$Procedure\ RIS(\pi, \pi^R, f(\pi))$

$Counter = 1$
$pos = 1, pt1 = -1,\ pt2 = -1$
$while(Counter \le n)do$
    $k = 1$
    $while\ (\pi_k\ != \ \pi^R_{pos})\ k = k + 1; endwhile$
    $pos = (pos + 1)(mod)n$
    $\pi_1 = remove\ job\ \pi^R_k\ \ from\ \pi$
    $update\ D_{j,k}\ matrix,\ \ i = 1,..,n - 1; k = 1.,,.m$
    $for\ \ i = 0\ \ to\ \ n$
        $\pi_2 = Insert(\pi_1, i, k)$
        $if\ \left(f_{fast}(\pi_2, i) < f(\pi)\right) then$
            $pt1 = k$
            $pt2 = i$
            $f(\pi) = f(\pi_2)$
    $endfor$
    $if\ (pt1 > -1)\ do$
        $\pi = insert\ job\ \pi_{pt1}\ at\ position\ pt2\ of\ \pi_1$
        $Counter = 1$
    $else$
        $Counter = Counter + 1$
    $endif$
$endwhile$
$return\ \pi\ and\ f(\pi)$
$endprocedure$

Figure 7. Referenced insertion scheme (RIS) local search.

The RIS local search is an insertion local search neighborhood. The details can be found in [47,54–59]. Note that the fast fitness calculation is used in the RIS local search.

$Procedure\ RSS(\pi, \pi^R, f(\pi))$

$Counter = 1$
$pos = 1, pt1 = -1, pt2 = -1$
$while(Counter \le n)do$
    $k = 1$
    $while\ (\pi_k\ != \ \pi^R_{pos})\ k = k + 1; endwhile$
    $pos = (pos + 1)(mod)n$
    $update\ D_{j,k}\ matrix,\ \ i = 1,..,n; k = 1.,,.m$
    $for\ \ i = 0\ \ to\ \ n$
        $\pi_2 = swap(\pi, i, k)$
        $if\ \left(f_{fast}(\pi_2, i) < f(\pi)\right) then$
            $pt1 = k$
            $pt2 = i$
            $f(\pi) = f(\pi_2)$
    $endfor$
    $if\ (pt1 > -1)\ do$
        $\pi =$ swap $job\ \pi_{pt1}$ with the one $at\ position\ pt2\ of\ \pi$
        $Counter = 1$
    $else$
        $Counter = Counter + 1$
    $endif$
$endwhile$
$return\ \pi\ and\ f(\pi)$
$endprocedure$

Figure 8. Referenced swap scheme (RSS) local search.

As seen in Figure 8, the RSS local search designates the job to be swapped by using the reference sequence π^R as in the RIS algorithm. Then, for each job π_k, it exchanges job π_k with each π_i until the last job in the sequence. As long as the sequence improves after a number n of swap moves, the counter is fixed to 1 so that the search starts from the beginning again. Note that the fast fitness calculation is used in RSS local search, too.

After the local search phase, it should be decided if the new sequence is accepted as the incumbent sequence for the next iteration. A simple simulated annealing type of acceptance criterion is used with a constant temperature, which is suggested by Osman and Potts [60], as follows:

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

where τP is a parameter to be adjusted.

4. Parameter Tuning

In this section, we first determine the parameters of the PFT_NEH(x) heuristic through a design of experiment (DOE) approach [61]. Then, we again make a design of experiment for the proposed VBIH algorithm in order to determine its parameters.

4.1. Parameter Tuning of PFT_NEX(x) Heuristic

As mentioned before, since the number x of sequences generated are not so costly until $n \leq 200$ in terms of CPU times, we determined it as $if \ (n \leq 200) \ then \ x = n$ else $x = 20$. Then, PFT_NEX(x) heuristic has two important parameters. Namely, μ and δ. To determine μ and δ parameters, we carry out a design of experiments (DOE) [61]. To do it, we generate random instances with the method proposed in [20]. In other words, random instances are generated for each combination of $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$. Five instances are generated for each job and machine combination, respectively. Ultimately, we obtained 75 instances for each pair of job and machine size. We consider two parameters, namely, the weight of the cost function μ and the size of the partial NEH heuristic δ. We have taken μ with 21 levels as $\mu = 0.00,\ 0.05,\ ..,1.00$ and δ with 10 levels as $\delta = 5,\ 10,\ 15,\ 20,\ 25,\ 30,\ 35,\ 40,\ 45,\ 50$. We conducted a full factorial design of experiments resulting in $21 \times 10 = 210$ treatments. For each job and machine combination, each instance is run for 210 treatments (210 PFT_ NEX(x) heuristics with μ and δ values). The relative percent deviation is calculated as follows:

$$RPD = \sum_{i=1}^{210}\left(\frac{C_i - C_{min}}{C_{min}}\right) * 100. \tag{11}$$

where C_i is the total flowtime generated by each treatment and C_{min} is the minimum total flowtime found amongst 210 treatments. This is repeated for five instances and RPD values are averaged for each treatment. Then the response variable is obtained by averaging the RPD values of 15 different job and machine combinations for each treatment. The DOE is coded in Visual C++13 and carried out on an Intel(R) Core(TM) i7-2600 CPU with 3.40 GHz PC with 8.00 GB memory.

Once the response variable was determined for each treatment, we analyze the main effects plot of parameters, which is given in Figure 9. Figure 9 suggests that μ (parW) and δ (parNEH) should be taken as 0.35 and 15, respectively.

However, Figure 9 suggests that μ values with 0.25, 0.30, 0.35, 0.40, 0.55 and 0.60 with $\delta = 15$ generate very good and diversified results on random benchmark instances. Table 1 summarizes the computational results of those μ values as well as the CPU times in seconds. It can be seen in Table 1 that $\mu = 0.30$ and $\mu = 0.35$ provided the best results as Figure 9 suggested. However, $\mu = 0.25,\ 0.40,\ 0.55$ *and* 0.60 provide some diversified solutions, too. Note that the PFT_ NEH(x) heuristics are substantially better than the HPF2 heuristic presented in [41] since it generated the ARPDs of 1.473 and 1.468, respectively when compared to the ARPD of 3.287. In addition, the same authors proposed GRASP-based algorithms as well as NHPF1 and NHPF2 heuristics for the initial solution procedures in [40]. PFT_ NEH(x) heuristics are also substantially better than NHPF1 and NHPF2 heuristics since the ARPDs of 1.473 percent and 1.468 percent are much better than those ARPDs of 3.018 percent and 2.797 percent. In addition to above, PFT_ NEH(x) heuristics were able to further improve some larger instances in about 0.760 s on overall average.

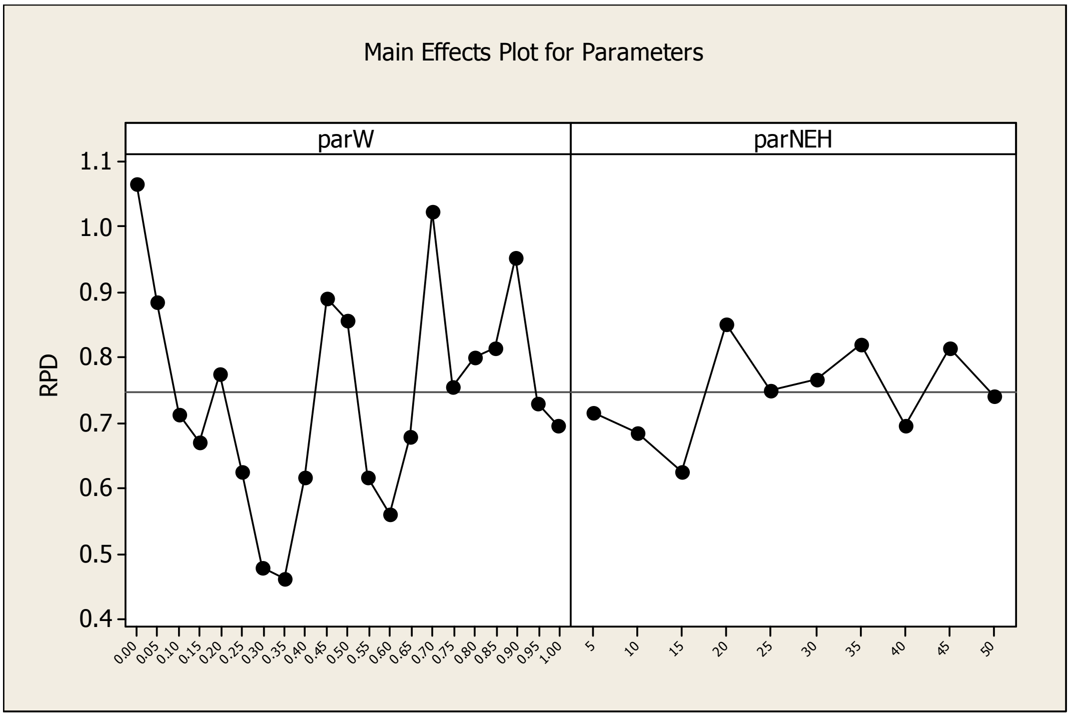

Figure 9. Main effects plot.

4.2. Parameter Tuning of VBIH Algorithm

In this section, we again present a DOE approach for parameter setting of the VBIH algorithms. In order to carry out experiments, we generate random instances with the method proposed in [20]. In other words, random instances are generated for each combination of $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$. Five instances are generated for each job and machine combination, respectively. Ultimately, we obtained 75 instances for each pair of job and machine size. We consider three parameters in the DOE approach. These are maximum block size (bS_{max}), maximum block insertion move size (mS_{max}); and temperature adjustment parameter (τP). We have taken the maximum block size with four levels as $bS_{max} \in (4, 8, 12, 16)$; the maximum block insertion move size with three levels as $mS_{max} \in (0.1 * (n - bS_{max})\,,\ 0.2 * (n - bS_{max})\,,\ 0.3 * (n - bS_{max}))$; and the temperature adjustment parameter with five levels as $\tau P \in (0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5)$. We conducted a full factorial design of experiments resulting in $4 \times 3 \times 5 = 60$ treatments. For each job and machine combination, each instance is run for 60 treatments with a maximum CPU time equal to $T_{max} = 10 \times n \times m$ milliseconds. The relative percent deviation is calculated as follows:

$$RPD = \sum_{i=1}^{60} \left(\frac{C_i - C_{min}}{C_{min}} \right) * 100. \tag{12}$$

where C_i is the total flowtime generated by each treatment and C_{min} is the minimum total flowtime found amongst 60 treatments. This is repeated for five instances and RPD values are averaged for each treatment. Then the response variable is obtained by averaging the RPD values of 15 different job and machine combinations for each treatment. The DOE is coded in Visual C++13 and carried out on an Intel (R) Core (TM) i7-2600 CPU with 3.40 GHz PC with 8.00 GB memory.

Table 1. Average relative percentage deviations for PFT_NEH(x) heuristics.

				μ Values and CPU (s)											
n × m	**HPF2 [41]**	**NHPF1 [40]**	**NHPF2 [40]**	**0.25**	**CPU (s)**	**0.30**	**CPU (s)**	**0.35**	**CPU (s)**	**0.40**	**CPU (s)**	**0.55**	**CPU (s)**	**0.60**	**CPU (s)**
20 × 5	4.038	2.928	3.059	1.368	0.002	1.190	0.000	1.158	0.003	1.014	0.003	1.264	0.000	1.190	0.002
20 × 10	3.156	2.719	2.340	1.069	0.000	1.228	0.003	1.005	0.002	1.154	0.000	1.175	0.000	1.005	0.003
20 × 20	3.989	2.857	2.766	1.011	0.002	0.942	0.003	1.016	0.003	1.063	0.003	1.009	0.002	1.047	0.003
50 × 5	3.929	3.903	3.528	3.029	0.022	2.775	0.023	2.774	0.024	3.204	0.025	2.881	0.022	2.915	0.025
50 × 10	3.664	4.191	3.665	2.718	0.039	2.651	0.039	2.608	0.041	2.671	0.039	2.655	0.041	2.871	0.041
50 × 20	5.318	4.398	4.237	2.453	0.078	2.409	0.077	2.274	0.073	2.310	0.075	2.210	0.080	2.258	0.078
100 × 5	3.816	3.757	3.668	2.989	0.116	2.743	0.114	2.702	0.125	2.692	0.116	2.822	0.116	2.913	0.120
100 × 10	4.087	4.450	3.964	2.611	0.211	2.783	0.216	2.598	0.213	2.569	0.213	2.755	0.223	2.863	0.217
100 × 20	5.554	4.428	4.539	2.223	0.431	2.030	0.434	2.125	0.433	2.064	0.433	2.353	0.434	2.207	0.436
200 × 10	2.362	2.505	1.915	1.057	1.730	0.944	1.740	1.053	1.737	1.070	1.736	1.160	1.731	1.336	1.736
200 × 20	2.811	2.676	2.478	0.777	3.553	0.669	3.580	0.889	3.594	0.780	3.552	1.000	3.552	1.122	3.555
500 × 20	1.595	1.464	1.533	−0.177	2.352	−0.158	2.402	−0.111	2.358	−0.082	2.349	0.066	2.350	0.048	2.352
200 × 5	2.394	2.260	1.936	1.400	0.917	1.292	0.920	1.264	0.917	1.314	0.919	1.727	0.917	1.833	0.920
500 × 5	1.191	1.330	1.027	0.545	0.755	0.356	0.758	0.317	0.750	0.475	0.775	0.825	0.761	0.916	0.758
500 × 10	1.398	1.399	1.307	0.272	1.194	0.247	1.195	0.353	1.189	0.248	1.188	0.477	1.191	0.564	1.191
Average	3.287	3.018	2.797	1.556	0.760	**1.473**	**0.767**	**1.468**	**0.764**	1.503	0.762	1.625	0.761	1.673	0.762

Bold: better results.

Once the response variable was determined for each treatment, we analyze the main effects plots of the parameters, which are given in Figure 10. Figure 10 suggests that the bS_{max} should be taken as $bS_{max} = 16$; mS_{max} should be taken as $mS_{max} = 0.1 * (n - bS_{max})$; and τP should be taken as $\tau P = 0.2$.

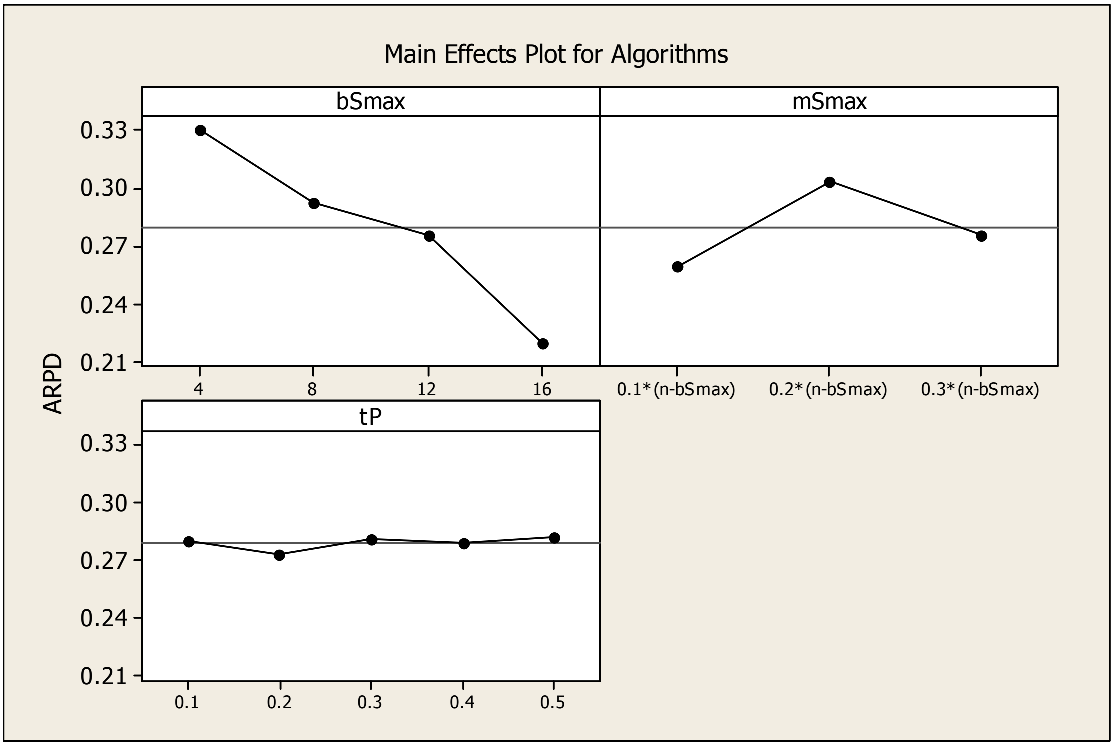

Figure 10. Main Effects Plots of Parameters.

The main effects plot might not be meaningful when there are significant interactions between the parameters. For this reason, the ANOVA table should be analyzed to see whether interactions are significant or not. The ANOVA results are given in Table 2. From Table 2, it can be seen that $bS_{max} * mS_{max}$ interaction is found to be significant because of the very high magnitude of F ratio and the p-value being less than α = 0.05 level. For these reasons, we look at the $bS_{max} * mS_{max}$ interaction plot, which is given in Figure 11.

As seen in Figure 11, $bS_{max} = 16$ with $mS_{max} = 0.3 * (n - bS_{max})$ generated the lowest ARPD. For this reason, we decided to take the parameters as follows: $bS_{max} = 16$; $mS_{max} = 0.3 * (n - bS_{max})$; and $\tau P = 0.2$.

Table 2. ANOVA table.

Source	*DF*	*Seq SS*	*Adj SS*	*Adj MS*	*F*	*p*
bS_{max}	3	0.095370	0.095370	0.031790	98.55	0.00
mS_{max}	2	0.019873	0.019873	0.009937	30.80	0.00
tP	4	0.000676	0.000676	0.000169	0.52	0.72
$bS_{max} * mS_{max}$	6	0.039930	0.039930	0.006655	20.63	0.00
$bS_{max} * tP$	12	0.002395	0.002395	0.000200	0.62	0.81
$mS_{max} * tP$	8	0.001585	0.001585	0.000198	0.61	0.76
Error	24	0.007742	0.007742	0.000323	-	-
Total	59	0.167569	-	-	-	-

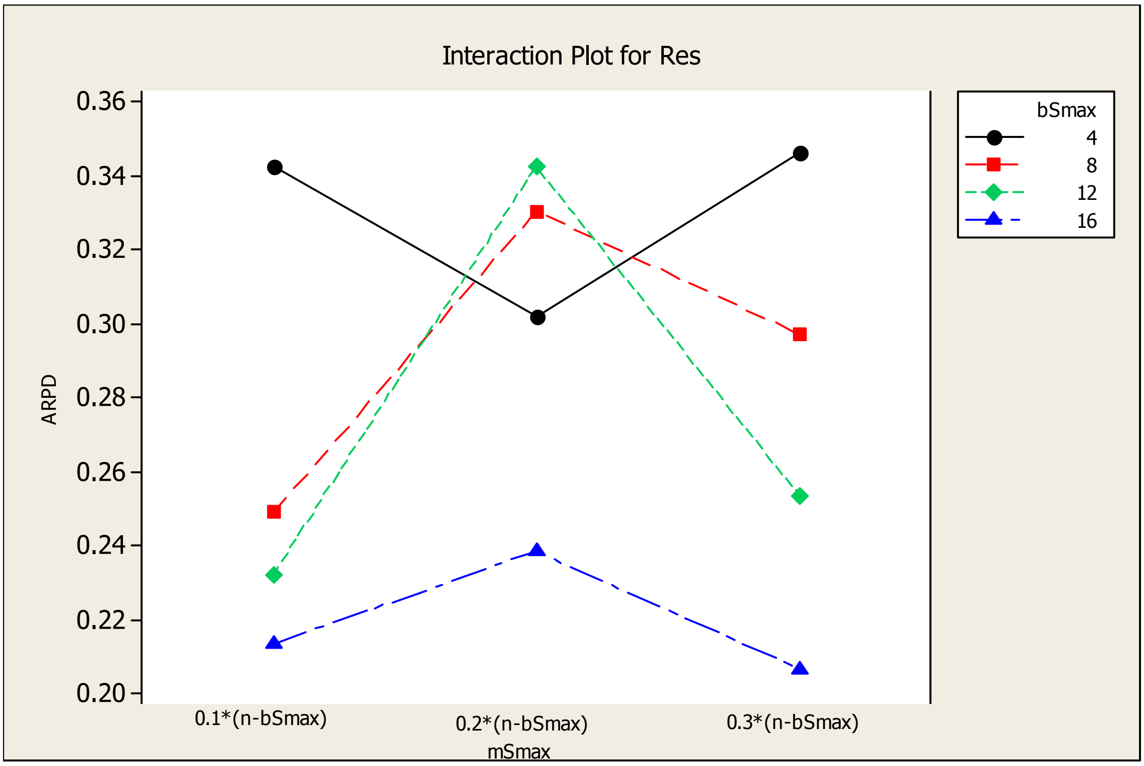

Figure 11. Interaction plot.

5. Computational Results

To test the performance of the VBIH algorithms proposed, extensive experimental evaluations and comparisons with other powerful methods are provided based on the well-known flowshop benchmark suite of Taillard [20]. The benchmark set is composed of 15 groups of the given problems with the size ranging from 20 jobs and 5 machines to 500 jobs and 20 machines, and each group consists of ten instances. However, in [62], these benchmark problems are extended to 200 $\times$ 5, 500 $\times$ 5 and 500 $\times$ 10 sizes with each containing 10 instances. Ultimately, we employ 150 instances as in the DABC_RCT algorithm in [41]. We treat them as blocking flow shop scheduling problems with the total flowtime criterion. In the experimental tests, all algorithms are coded in Visual C++13 and carried out on an Intel(R) Core(TM) i7-2600 CPU with 3.40 GHz PC with 8.00GB memory. Note that the maximum CPU time is fixed at $T_{max} = 100 \times n \times m$ milliseconds for all algorithms compared. We compare to the following best performing algorithms from the literature:

1. **DABC_RCT in [41].** The DABC_RCT algorithm is a very efficient algorithm and has three phases. In the employed bee phase, the TNO procedure is employed with the VLS local search. In the onlooker bee phase, the path-relinking approach is employed to generate the onlooker bees. In the scout bee phase, HPF2 is used to generate the scout bees. We refer to [41] for the details. We have coded the DABC_RCT algorithm in Visual C+13 to have a fair comparison. Note that the DABC_RCT algorithm uses HPF2 heuristic as an initial solution. Since our PFT_NEH(x) heuristic is substantially better than HPF2 heuristic, we employ the PFT_NEH(x) heuristic with $\mu = 0.35$ and $\delta = 15$ as one of the solution in the population. The rest of the population individuals are constructed randomly as suggested in the DABC_RCT algorithm and we denote it as the DABC*_RCT algorithm to have a fair comparison. The same parameters are also used which are suggested in the DABC_RCT algorithm [41]. Note that fast fitness calculation is employed to accelerate the insertion and swap neighborhood structures in the VLS local search they employed in the TNO procedure.

2. **IG_RIS algorithm in [47,54–59].** To be fair again, we employ PFT_NEH(x) heuristic with $\mu = 0.35$ and $\delta = 15$ as an initial solution in the IG_RIS algorithm. IG_RIS algorithm relies on the destruction and construction procedure, where dS number of jobs is removed from a solution and they are reinserted to the partial solution sequentially. Then RIS local search is applied to the solution obtained after destruction and construction procedure. Note that fast fitness calculation is employed to accelerate the RIS insertion local search as in this paper. It is also employed in the destruction and construction procedure, too.
3. **VBIH algorithms in this paper.** Since the PFT_NEH(x) heuristics provide very diversified initial solutions, we run the VBIH algorithm with μ values with 0.25, 0.30, 0.35, 0.40, 0.55 and 0.60 with $\delta = 15$. Then we denote them as VBIH1, VBIH2, VBIH3, VBIH4, VBIH5 and VBIH6. In addition, when the maximum block size is equal to 1, the VBIH algorithm becomes an iterated local search. In other words, the current solution is perturbed with several insertion moves and then the VLS local search is applied to the solution after perturbation. Then, the acceptance criterion is imposed to the solution obtained. We denote this variant of the VBIH algorithm as IVLS algorithm.

   Each instance is run for five (R) independent replications and the relative percentage deviation. RPD is computed as follows:

$$ARPD = \sum_{i=1}^{R} \left(\frac{H - BKS}{BKS} \right) * 100/R \tag{13}$$

where $H,\ BKS,\ and\ R$ are the total flowtime value generated by the algorithms in each run, the most recent best-known solution value reported in [41], and the number of runs, respectively. In addition, the ARPD for each instance, totally 150, is recorded to make statistical analyses.

The computational results are given in Table 3. The first observation from Table 3 that the IG_RIS and DABC_RCT algorithms are not competitive to the DABC*_RCT and VBIH variants. Especially, even though the fast fitness calculation is employed in the IG_RIS algorithm, which is known as one of the best algorithms in the scheduling literature, it could not be able to generate competitive results. In fact, its local search is based on only the insertion neighborhood, which is very effective for the makespan criterion. However, the results in Table 3 indicate that the swap neighborhood should be used in algorithms designed for the total flowtime criterion. The second observation is the performance of the DABC*_RCT algorithm, which is quite competitive to the VBIH variants. The overall ARPD is decreased from 0.593 percent to 0.376 percent due to the use of the PFT_NEH(x) heuristic as a solution in the initial population. It indicates that the proposed PFT_NEH(x) heuristic was so effective on the results of the DABC*_RCT algorithm. Amongst the VBIH variants, the first four variants were able to generate results ranging from 0.303 percent to 0.285 percent. As can be seen from Table 3, especially, the larger instances with sizes 500×20, 500×5 and 500×10 were further improved by the first four variants of the VBIH algorithms.

Table 3. Average relative percentage deviations for algorithms compared.

n × m	IG_RIS	DABC_RCT	IVLS	DABC*_RCT	VBIH1	VBIH2	VBIH3	VBIH4	VBIH5	VBIH6
20 × 5	0.049	0.004	0.068	0.006	0.001	0.000	0.000	0.000	0.000	0.000
20 × 10	0.016	0.021	0.151	0.031	0.000	0.000	0.000	0.000	0.000	0.000
20 × 20	0.010	0.015	0.032	0.002	0.000	0.000	0.000	0.000	0.000	0.000
50 × 5	0.901	0.788	0.713	0.490	0.418	0.354	0.346	0.376	0.296	0.325
50 × 10	0.819	0.752	0.916	0.752	0.562	0.455	0.457	0.424	0.463	0.502
50 × 20	0.521	0.561	0.652	0.490	0.318	0.378	0.316	0.355	0.330	0.321
100 × 5	1.555	1.135	1.133	0.937	0.650	0.764	0.709	0.650	0.696	0.714
100 × 10	1.639	1.301	1.358	1.285	1.070	1.179	1.126	1.098	1.112	1.118
100 × 20	1.147	1.279	0.927	0.983	0.759	0.833	0.799	0.903	0.796	0.750
200 × 10	0.633	0.559	0.236	0.312	0.427	0.243	0.377	0.293	0.297	0.308
200 × 20	0.337	0.581	0.193	0.201	0.146	0.227	0.211	0.255	0.430	0.421
500 × 20	−0.229	0.426	−0.381	−0.274	−0.353	−0.374	−0.395	−0.303	−0.192	−0.185
200 × 5	0.854	0.470	0.373	0.358	0.392	0.325	0.341	0.372	0.388	0.401
500 × 5	0.259	0.397	−0.014	−0.022	0.150	−0.012	−0.053	−0.042	0.299	0.369
500 × 10	0.234	0.607	−0.033	0.092	0.000	−0.031	0.035	−0.046	0.174	0.282
Average	0.583	0.593	0.422	0.376	0.303	**0.289**	**0.285**	**0.289**	0.339	0.355

Bold: better results.

In order to see the statistical difference between algorithms, we provide the interval plot of the algorithms compared in Figure 12. Since the 95% confidence intervals of DABC*_RCT and VBIH variants do not overlap, we can conclude that the DABC*_RCT and the VBIH variants generated results, which are statistically significant to those results generated by the IG_RIS and DABC_RCT algorithms. When we look at the confidence intervals of the DABC*_RCT and VBIH variants, there are overlaps between the algorithms. However, an overlap does not mean that there is no difference. There may be statistically significant difference even if there is an overlap [63]. To determine the difference, the paired *t*-tests should be used [63].

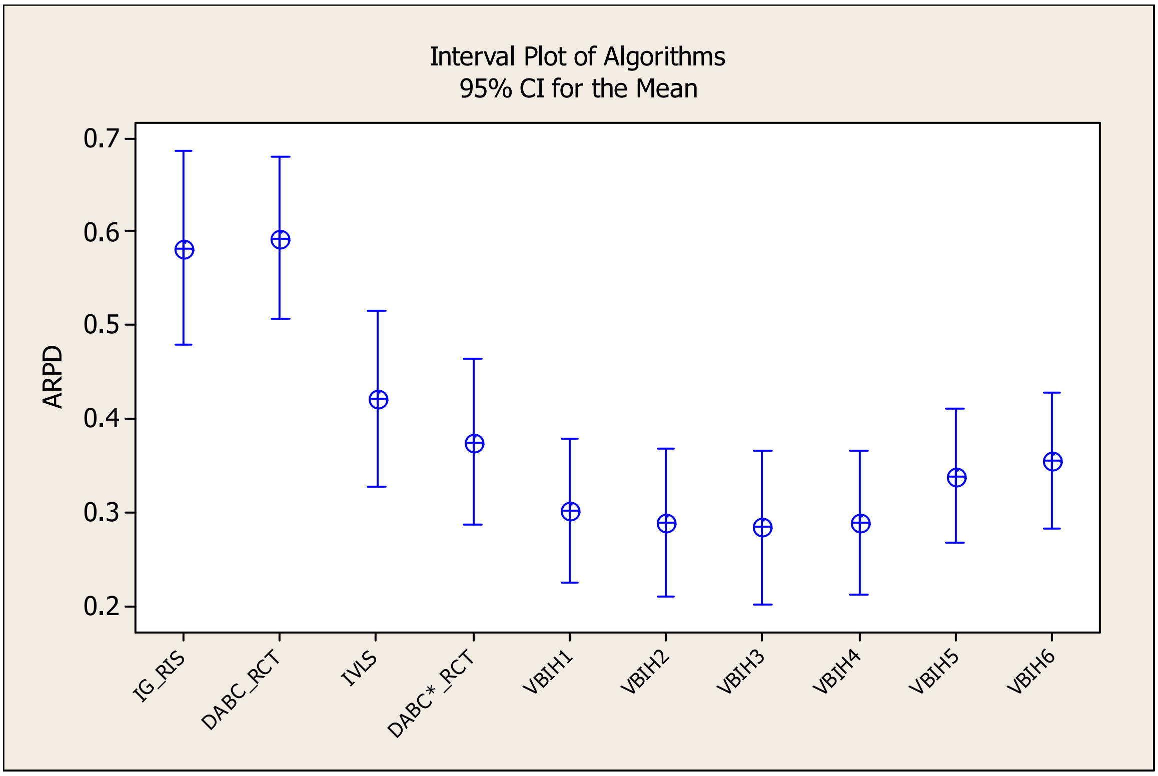

Figure 12. Interval plot of algorithms compared.

The results of the paired *t*-tests are given in Table 4. If the 95% confidence interval for the mean difference between the two compared algorithms does not include zero values, it indicates that there is a difference between the two algorithms compared. In addition, *p*-values, which are smaller than $\alpha = 0.05$ level, further suggest that the compare algorithms perform differently. As can be seen from Table 4, the IVLS, VBIH1, VBIH2, VBIH3 and VBIH4 algorithms are statistically better than the DABC*_RCT algorithm. However, the VBIH5 and VBIH 6 algorithms are statistically equivalent.

Table 4. Paired *t*-test for variable block insertion heuristic (VBIH) variants versus discrete artificial bee colony (DABC)*_RCT algorithm.

Algorithm vs. algorithm	95% CI for Mean Difference	*p*-Value
IVLS − DABC*_RCT	(0.0041, 0.0875)	0.032
VBIH1 − DABC*_RCT	(−0.1162, −0.0298)	0.001
VBIH2 − DABC*_RCT	(−0.1242, −0.0486)	0.000
VBIH3 − DABC*_RCT	(−0.1193, −0.0639)	0.000
VBIH4 − DABC*_RCT	(−0.1258, −0.0474)	0.000
VBIH5 − DABC*_RCT	(−0.0843, 0.0117)	0.137
VBIH6 − DABC*_RCT	(−0.0671, 0.0252)	0.371

Finally, in Table 5, we provide the best solutions found by each algorithm. 52 out of 150 problems instances are further improved together with 15 solutions being equal during in this study. All the results and permutations are available based on request.

Table 5. Best-known solutions.

BKS [41]	IVLS	VBIH1	VBIH2	VBIH3	VBIH4	VBIH5	VBIH6	DABC_RCT	DABC*_RCT
Problem Set 20×5									
14953	***14953***	***14953***	***14953***	***14953***	***14953***	***14953***	***14953***	***14953***	***14953***
16343	16349	***16343***	***16343***	***16343***	***16343***	***16343***	***16343***	***16343***	***16343***
14297	***14297***	***14297***	***14297***	***14297***	***14297***	***14297***	***14297***	***14297***	***14297***
16483	***16483***	***16483***	***16483***	***16483***	***16483***	***16483***	***16483***	***16483***	***16483***
14212	***14212***	***14212***	***14212***	***14212***	***14212***	***14212***	***14212***	***14212***	***14212***
14624	***14624***	***14624***	***14624***	***14624***	***14624***	***14624***	***14624***	***14624***	***14624***
14936	14938	***14936***	***14936***	***14936***	***14936***	***14936***	***14936***	*14938*	*14938*
15193	15240	***15193***	***15193***	***15193***	***15193***	***15193***	***15193***	***15193***	***15193***
15544	***15544***	***15544***	***15544***	***15544***	***15544***	***15544***	***15544***	***15544***	***15544***
14392	***14392***	***14392***	***14392***	***14392***	***14392***	***14392***	***14392***	***14392***	***14392***
Problem Set 20×10									
22358	22537	***22358***	***22358***	***22358***	***22358***	***22358***	***22358***	***22358***	***22358***
23881	***23881***	***23881***	***23881***	***23881***	***23881***	***23881***	***23881***	***23881***	***23881***
20873	***20873***	***20873***	***20873***	***20873***	***20873***	***20873***	***20873***	***20873***	***20873***
19916	20020	***19916***	***19916***	***19916***	***19916***	***19916***	***19916***	***19916***	***19916***
20196	***20196***	***20196***	***20196***	***20196***	***20196***	***20196***	***20196***	***20196***	***20196***
20126	***20126***	***20126***	***20126***	***20126***	***20126***	***20126***	***20126***	***20126***	***20126***
19471	***19471***	***19471***	***19471***	***19471***	***19471***	***19471***	***19471***	***19471***	***19471***
21330	21369	***21330***	***21330***	***21330***	***21330***	***21330***	***21330***	***21330***	***21330***
21585	***21585***	***21585***	***21585***	***21585***	***21585***	***21585***	***21585***	***21585***	***21585***
22582	***22582***	***22582***	***22582***	***22582***	***22582***	***22582***	***22582***	***22582***	***22582***
Problem Set 20×20									
34683	***34683***	***34683***	***34683***	***34683***	***34683***	***34683***	***34683***	***34683***	***34683***
32855	***32855***	***32855***	***32855***	***32855***	***32855***	***32855***	***32855***	***32855***	***32855***
34825	***34825***	***34825***	***34825***	***34825***	***34825***	***34825***	***34825***	***34825***	***34825***
33006	***33006***	***33006***	***33006***	***33006***	***33006***	***33006***	***33006***	***33006***	***33006***
35328	***35328***	***35328***	***35328***	***35328***	***35328***	***35328***	***35328***	***35328***	***35328***
33720	***33720***	***33720***	***33720***	***33720***	***33720***	***33720***	***33720***	***33720***	***33720***
33992	***33992***	***33992***	***33992***	***33992***	***33992***	***33992***	***33992***	***33992***	***33992***
33388	***33388***	***33388***	***33388***	***33388***	***33388***	***33388***	***33388***	***33388***	***33388***
34798	***34798***	***34798***	***34798***	***34798***	***34798***	***34798***	***34798***	***34798***	***34798***
33174	***33174***	***33174***	***33174***	***33174***	***33174***	***33174***	***33174***	***33174***	***33174***
Problem Set 50×5									
72672	72758	72672	72696	***72672***	72696	72758	72827	73135	72768
78140	78707	78254	78332	78181	78181	78181	78284	78327	78295
72913	73211	73096	73224	73101	73224	***72913***	72994	***72913***	73224
77399	77711	77513	77571	77547	77586	77547	77547	77582	77607
78353	78705	78627	78579	78544	78363	78511	78511	78767	78620
75402	***75402***	75661	75606	75475	75593	***75402***	75514	76122	75615
73842	74322	73952	74202	73952	73890	73952	73891	73954	73890
73442	73964	73945	***73442***	73834	***73442***	73549	73549	73858	***73442***
70871	71360	70905	***70871***	***70871***	70883	70883	70883	71096	71105
78729	79271	78773	***78729***	***78729***	78807	78729	***78729***	78773	79093
Problem Set 50×10									
99674	100508	100373	***99674***	100299	100059	99721	100410	99900	100325
95608	95669	95907	96157	95669	96047	95876	95876	96565	96367
91791	92760	91956	92090	***91791***	92090	92090	92276	92588	92524
98454	98767	98475	98689	***98454***	***98454***	98507	98507	98692	98576
98164	98286	98243	***98164***	98230	***98164***	98228	98228	98610	98228
97246	97826	97637	97779	97431	97530	97558	97333	98029	97625
99953	100142	100030	99965	99965	***99953***	99971	100116	100440	100584
98027	98231	98149	98271	98476	98436	98543	98270	98723	98521
96708	97248	96708	96996	96996	***96708***	97142	***96708***	96978	97634
98019	99012	98316	98316	98316	98316	98053	98053	98316	98362

Table 5. *Cont.*

BKS [41]	IVLS	VBIH1	VBIH2	VBIH3	VBIH4	VBIH5	VBIH6	DABC_RCT	DABC*_RCT
Problem Set 50 × 20									
136865	137240	136881	137075	137075	136968	137005	137005	136958	137161
129958	130333	130115	129975	130292	130244	130248	129975	130176	130303
127617	128784	127617	127957	***127617***	128073	127617	***127617***	128033	128393
131889	132452	132270	132169	132270	132103	131943	131943	132283	132169
130967	131159	130979	131217	131233	***130967***	131196	130979	131351	131064
131760	132121	131985	131760	131760	131929	131926	131921	**131593**	132007
134217	134857	134222	134534	134572	134534	134726	134451	134715	134636
132990	133502	132990	133210	133210	133210	133309	133309	133526	133210
132599	132819	132757	132715	132901	132757	***132599***	***132599***	***132599***	133120
135710	136166	135985	136162	136224	136363	136248	136146	136483	136473
Problem Set 100 × 5									
288332	289888	288446	289216	288765	288854	290346	288904	289463	288807
280491	282897	281066	280743	280853	**280073**	280873	280929	282857	282563
276228	277904	**275863**	276229	276322	277751	276589	276695	278117	277968
259596	262968	261462	261867	261601	261985	261715	261231	263990	261478
273086	275571	274651	274335	274451	274690	274005	274817	274804	275092
267381	271534	269418	267899	270406	269506	270408	269194	268899	269356
274744	277150	275884	277009	277342	276656	275491	276609	277163	277535
269689	272776	271187	270939	270945	270774	270668	271001	271916	271719
284816	286683	285308	285238	284901	**284652**	284952	284755	287494	284856
282005	282659	282969	283292	282814	282366	282367	282719	283596	283939
Problem Set 100 × 10									
354083	357361	354586	355794	356308	354892	**353321**	354624	354570	356911
333379	335775	335738	336636	335905	335268	336469	336601	337164	336403
343957	346543	344337	345157	345524	345069	344824	345654	344863	345889
359259	362441	361621	363410	360537	361230	360709	359680	361328	364095
338537	339455	339573	339976	338941	341468	341261	340741	340771	341207
327254	331594	329327	328482	330769	328075	328693	329377	330296	329454
335366	339300	338219	337094	339001	338948	338091	338091	337997	338643
343174	346305	345286	344609	344905	346013	345217	343843	344417	344886
344563	357006	354676	357664	356781	356050	355165	354659	356177	356628
347845	349966	349546	348910	350290	349351	350879	350580	350674	350693
Problem Set 100 × 20									
425224	427753	427810	426032	427688	426582	427549	426845	427899	426093
435289	438146	436000	436495	437478	436409	437908	436205	439020	437380
430634	431419	432953	433663	431713	434502	431905	432067	435222	434241
432314	436203	435708	435999	435001	437256	435401	435880	437725	438501
426405	429524	428737	429044	428845	430388	428200	429321	429886	429472
430308	434216	432099	431680	432345	433041	432139	434523	434105	431202
436642	440171	440174	441664	438725	441037	437488	439062	440081	442024
440930	445900	443867	445170	445219	443812	443648	444357	444299	445169
432876	435264	434452	435075	434655	433631	434701	434806	437318	435536
437286	440819	440918	438439	440992	438621	440484	438530	443779	439693
Problem Set 200 × 10									
1281633	1281292	1286077	1283314	1281947	1281401	**1280745**	1282445	1285819	1281919
1283164	1279913	1285268	1279947	1280799	1280548	1280432	1281886	**1279240**	1280107
1277933	1280412	1281905	1279882	1282228	1282447	1281843	1284700	1282812	**1276690**
1271502	1275865	1280417	1280027	1274711	1273979	1274071	1277414	1278833	1280871
1275901	1282570	1276110	1286907	1279126	1280511	**1275710**	1279823	1277954	1285519
1251213	1252110	1258463	1248655	1252536	1254724	1252145	1248058	1258054	**1244667**
1304158	1307602	**1303545**	1311626	1305541	1305201	1306954	1302585	1309672	1309810
1298900	1296591	1301184	1303103	1297501	**1295844**	1302459	1299302	1296469	1302829
1277801	1270883	1278023	1273946	**1270118**	1277145	1278259	1277646	1278808	1272519
1273794	1281472	1279452	1284374	1281680	1278887	1282278	1282763	1280799	1283774

Table 5. *Cont.*

BKS [41]	IVLS	VBIH1	VBIH2	VBIH3	VBIH4	VBIH5	VBIH6	DABC_RCT	DABC*_RCT
Problem Set 200 × 20									
1499623	1505301	1509409	1506809	1503861	1508265	1516354	1511393	1512033	1508941
1541253	1538208	1540131	**1531179**	1538989	1538626	1538036	1536971	1538890	1534481
1546279	1546915	1553963	1557771	1550237	1553555	1558080	1558445	1559908	1558470
1540822	1542961	**1537852**	1545112	1544472	1544429	1548607	1547062	1541447	1544097
1514600	1515692	1514609	1520799	1515470	1517899	1520051	1521375	1518989	1520058
1528885	1535006	1533907	1532007	1535114	1534306	1533721	1533357	1543380	**1528532**
1532090	1535822	1534243	1532903	1536909	1534423	1540592	1540200	1539144	**1529204**
1543229	1537885	1540539	1542855	1540221	**1538906**	1541537	1542151	1543159	1543637
1524293	1522771	1517701	1516868	1522958	1525068	1524214	1519558	1524550	**1514130**
1535329	1533492	**1528995**	1534456	1534830	1535504	1538711	1536245	1540038	1534682
Problem Set 500 × 20									
8719682	8706102	8701976	8693435	8705502	**8691397**	8704588	8720565	8730777	8702888
8849228	8823698	**8804700**	8813687	8825577	8816790	8820512	8825084	8908220	8814543
8789777	8745145	8746129	8742781	**8728918**	8750421	8759822	8746826	8811943	8739388
8828454	**8791325**	8807711	8793936	8793088	8795242	8807649	8803254	8862426	8795162
8796337	**8735014**	8722394	8741122	8736988	8755137	8781374	8756332	8791516	8774397
8837577	8804938	8821880	8793126	8803399	8820331	8809748	8824192	8870293	**8791898**
8729909	**8718042**	8734168	8734722	8719324	8737033	8750115	8748411	8802463	8733839
8800506	8766044	8756165	8773136	8772308	8774052	**8761218**	8787988	8796375	8772084
8782791	8739864	8751027	**8727721**	8743741	8743940	8761953	8747533	8799104	8773353
8849551	8791671	8802420	**8780952**	8788594	8796622	8810047	8818606	8878995	8805877
Problem Set 200 × 5									
1071652	1073055	1072760	1076925	1071889	1073065	1073348	1071170	1071705	**1070790**
1026640	1026510	1021433	1025826	1025310	1023515	1028051	1024432	**1019431**	1025138
1059120	1064728	1062714	1064025	1064244	1060982	1065879	1064569	1062759	1061449
1044074	1048420	1051225	**1042391**	1048350	1049298	1042292	1047066	1048212	1044776
1064274	1064019	1064175	1060847	1064649	1061081	1060213	1062069	**1060420**	1062216
1021482	1024893	1027578	1029903	1025409	1027670	1029104	1030626	1026561	1029891
1082018	1081107	**1079945**	1082921	1081033	1081780	1083320	1084464	1083668	1081968
1043921	1047490	1048609	1050141	1045691	1048150	1051041	1048321	1050936	1049380
1057482	1057673	1058438	1056355	1058281	1058369	**1056063**	1055705	1058199	1059175
1037496	1043719	1043777	1039727	1042310	1045628	1039183	1043266	**1036938**	1039695
Problem Set 500 × 5									
6389122	6375325	6381517	6371100	6371117	**6366762**	6397170	6387506	6403589	6369864
6415066	6413469	6433713	**6392620**	6400361	6396807	6432279	6436552	6421048	6392856
6460745	6426771	**6426591**	6435399	6434882	6434628	6440953	6454047	6478507	6430821
6334201	**6303859**	6323236	6305175	6306089	6318146	6337465	6334555	6364065	6318682
6373873	6383164	6392640	6369007	6383774	**6355801**	6408507	6413732	6397351	6369219
6282522	6275452	6281428	6283594	6277932	**6274362**	6292695	6301826	6302507	6283635
6244926	6262136	6262957	6262423	6260089	6261444	6285376	6273743	6261620	6258574
6352627	6367281	6395544	6370417	6370566	6377367	6376438	6392365	**6350755**	6366156
6328390	6335967	6342425	6336154	6335220	6343154	6342796	6359365	6354617	6330549
6309180	6309639	6314591	**6297997**	6307224	6300714	6320912	6323148	6346828	6307074
Problem Set 500 × 10									
7552404	**7514159**	7534854	7522904	7519846	7523259	7533739	7549934	7577869	7541901
7665025	7632382	7633377	7649655	7635561	**7622243**	7642501	7652010	7658541	7643615
7626599	7590037	7599850	**7580415**	7588202	7590780	7622675	7622445	7652497	7603273
7626405	7618385	7600996	7633161	7619058	**7615308**	7645654	7623706	7635679	7635154
7479900	7484025	7468087	7472600	7468703	7478446	**7464923**	7496738	7504574	7476013
7537299	7548071	7546071	7563456	7551273	7551039	7572887	7566574	7586150	7562912
7510712	7505921	7502693	7490848	7504096	7482595	**7478959**	7514666	7534649	7491561
7562013	7577902	7599263	7598437	7588036	7598947	7599345	7635525	7607737	7599718
7550242	7538219	7537118	**7533874**	7539127	7547730	7577922	7574227	7581486	7536618
7549596	7577156	7596351	7588889	7580750	7562898	7611269	7589683	7662823	7594287

Bold: better results, Bold Italic: equal results.

6. Conclusions

This paper presents a VBIH algorithm to solve the blocking flowshop scheduling problem with the total flowtime criterion. To the best of our knowledge, this is the first reported application of the VHIH algorithm to the blocking flowshop scheduling problem with the total flowtime criterion. Once an initial solution is constructed, the VBIH algorithm begins with a minimum block size of one. It removes the block from the current sequence and inserts the block into the partial sequence with a predetermined move size. Then, a variable local search (VLS) is applied to solution obtained after several block insertions. As long as the solution improves, it keeps the same block size. However, if the solution does not improve, the block size is increased by one. This process is repeated until the block size reaches at the maximum block size. In addition, we present a novel constructive heuristic based on profile fitting heuristic from the literature with results improving especially some larger instances in a few seconds. Parameters of the constructive heuristic and the VBIH algorithm are determined through a design of experiment approach. Extensive computational results on Taillard's well-known benchmark suite show that the proposed VBIH algorithm outperforms the discrete artificial bee colony algorithm, which has recently been proposed with the new best known solutions. Ultimately, 52 out of the 150 best known solutions are further improved with substantial margins.

Supplementary Materials: The supplementary materials are available online at http://www.mdpi.com/1999-4893/9/4/71/s1.

Author Contributions: M. Fatih Tasgetiren and Quan-Ke Pan and Damla Kizilay have written the C++ codes. Damla Kizilay and Gao Kaizhou have carried out all the DOE runs and prepared the analyses of DOE runs. M. Fatih Tasgetiren and Quan-Ke Pan have written the paper. All authors have read and approved the final manuscript.

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

References

1. Błażewicz, J.; Ecker, K.H.; Pesch, E.; Schmidt, G.; Weglarz, J. *Handbook on Scheduling: From Theory to Applications*; Springer: Berlin/Heidelberg, Germany, 2007.
2. Pan, Q.K.; Ruiz, R. An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. *Omega* **2012**, *40*, 166–180. [CrossRef]
3. Vallada, E.; Ruiz, R. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega* **2010**, *38*, 57–67. [CrossRef]
4. Ruiz-Torres, A.J.; Ho, T.J.; Ablanedo-Rosas, J.H. Makespan and workstation utilization minimization in a flowshop with operations flexibility. *Omega* **2011**, *39*, 273–282. [CrossRef]
5. Grabowski, J.; Pempera, J. Sequencing of jobs in some production system. *Eur. J. Oper. Res.* **2000**, *125*, 535–550. [CrossRef]
6. Ronconi, D.P. A note on constructive heuristics for the flowshop problem with blocking. *Int. J. Prod. Econ.* **2004**, *87*, 39–48. [CrossRef]
7. Hall, N.G.; Sriskandarajah, C. A survey of machine scheduling problems with blocking and no-wait in process. *Oper. Res.* **1996**, *44*, 510–525. [CrossRef]
8. Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Rinnooy Kan, A.H.G. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discret. Math.* **1979**, *5*, 287–362.
9. Gilmore, P.C.; Lawler, E.L.; Shmoys, D.B. Well-solved special cases. In *The Traveling Salesman Problem: A Guided* Tour of Combinatorial Optimization; Lawler, E.L., Lenstra, K.L., Rinooy Kan, A.H.G., Shmoys, D.B., Eds.; John Wiley & Sons: Hoboken, NJ, USA, 1985; pp. 87–143.
10. McCormick, S.T.; Pinedo, M.L.; Shenker, S.; Wolf, B. Sequencing in an assembly line with blocking to minimize cycle time. *Oper. Res.* **1989**, *37*, 925–936. [CrossRef]
11. Leisten, R. Flowshop sequencing problems with limited buffer storage. *Int. J. Prod. Res.* **1990**, *28*, 2085–2100. [CrossRef]
12. Nawaz, M.; Enscore, E.E.J.; Ham, I. A heuristic algorithm for the *m*-machine, *n*-job flow shop sequencing problem. *Omega* **1983**, *11*, 91–95. [CrossRef]

13. Ronconi, D.P.; Armentano, V.A. Lower bounding schemes for flowshops with blocking in-process. *J. Oper. Res. Soc.* **2001**, *52*, 1289–1297. [CrossRef]
14. Abadi, I.N.K.; Hall, N.G.; Sriskandarajh, C. Minimizing cycle time in a blocking flowshop. *Oper. Res.* **2000**, *48*, 177–180. [CrossRef]
15. Ronconi, D.P.; Henriques, L.R.S. Some heuristic algorithms for total tardiness minimization in a flowshop with blocking. *Omega* **2009**, *37*, 272–281. [CrossRef]
16. Pan, Q.K.; Wang, L. Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. *Omega* **2012**, *40*, 218–229. [CrossRef]
17. Liu, J.Y.; Reeves, C.R. Constructive and composite heuristic solutions to the $P//\sum Ci$ scheduling problem. *Eur. J. Oper. Res.* **2001**, *132*, 439–452. [CrossRef]
18. Caraffa, V.; Ianes, S.; Bagchi, T.P.; Sriskandarajah, C. Minimizing makespan in a blocking flowshop using genetic algorithms. *Int. J. Prod. Econ.* **2001**, *70*, 101–115. [CrossRef]
19. Ronconi, D.P. A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking. *Ann. Oper. Res.* **2005**, *138*, 53–65. [CrossRef]
20. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [CrossRef]
21. Grabowski, J.; Pempera, J. The permutation flow shop problem with blocking. A tabu search approach. *Omega* **2007**, *35*, 302–311. [CrossRef]
22. Wang, L.; Zhang, L.; Zheng, D. An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Comput. Oper. Res.* **2006**, *33*, 2960–2971. [CrossRef]
23. Liu, B.; Wang, L.; Jin, Y. An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. *Comput. Oper. Res.* **2008**, *35*, 2791–2806. [CrossRef]
24. Qian, B.; Wang, L.; Huang, D.X.; Wang, X. An effective hybrid DE-based algorithm for flow shop scheduling with limited buffers. *Int. J. Prod. Res.* **2009**, *47*, 1–24. [CrossRef]
25. Liang, J.J.; Pan, Q.; Tiejun, C.; Wang, L. Solving the blocking flow shop scheduling problem by a dynamic multi-swarm particle swarm optimizer. *Int. J. Adv. Manuf. Technol.* **2011**, *55*, 755–762. [CrossRef]
26. Wang, L.; Pan, Q.K.; Tasgetiren, M.F. A Hybrid Harmony Search Algorithm for the Blocking Permutation Flow Shop Scheduling Problem. *Comput. Ind. Eng.* **2011**, *61*, 76–83. [CrossRef]
27. Wang, L.; Pan, Q.K.; Suganthan, P.N.; Wang, W.H.; Wang, Y.M. A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Comput. Oper. Res.* **2010**, *37*, 509–520. [CrossRef]
28. Ribas, I.; Companys, R.; Tort-Martorell, X. An iterated greedy algorithm for the flowshop scheduling with blocking. *Omega* **2011**, *39*, 293–301. [CrossRef]
29. Wang, C.; Song, S.; Gupta, J.N.D.; Wu, C. A three-phase algorithm for flowshop scheduling with blocking to minimize makespan. *Comput. Oper. Res.* **2012**, *39*, 2880–2887. [CrossRef]
30. Lin, S.W.; Ying, K.C. Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm. *Omega* **2013**, *41*, 383–389. [CrossRef]
31. Pan, Q.K.; Wang, L.; Sang, H.Y.; Li, J.Q.; Liu, M. A high performing memetic algorithm for the flowshop scheduling problem with blocking. *IEEE Trans. Autom. Sci. Eng.* **2013**, *10*, 741–756.
32. Ribas, I.; Companys, R.; Tort-Martorell, X. A competitive variable neighbourhood search algorithm for the blocking flow shop problem. *Eur. J. Ind. Eng.* **2013**, *7*, 729–754. [CrossRef]
33. Ding, J.-Y.; Song, S.; Gupta, N.D.J.; Wang, C.; Zhang, R.; Wu, C. New block properties for flowshop scheduling with blocking and their application in an iterated greedy algorithm. *Int. J. Prod. Res.* **2015**, *54*, 4759–4772. [CrossRef]
34. Armentano, V.A.; Ronconi, D.P. Minimização do tempo total de atraso no problema de flowshop com buffer zero através de busca tabu. *Gestao Produçao* **2000**, *7*, 352–362. (In Portuguese) [CrossRef]
35. Ribas, I.; Companys, R.; Tort-Martorell, X. An efficient iterated local search algorithm for the total tardiness blocking flow shop problem. *Int. J. Prod. Res.* **2013**, *51*, 5238–5252. [CrossRef]
36. Wang, L.; Pan, Q.K.; Tasgetiren, M.F. Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms. *Expert Syst. Appl.* **2010**, *37*, 7929–7936. [CrossRef]
37. Deng, G.; Xu, Z.; Gu, X. A discrete artificial bee colony algorithm for minimizing the total flow time in the blocking flow shop scheduling. *Chin. J. Chem. Eng.* **2012**, *20*, 1067–1073. [CrossRef]
38. Khorasanian, D.; Moslehi, G. An iterated greedy algorithm for solving the blocking flowshop scheduling problem with total flow time criteria. *Int. J. Ind. Eng. Prod. Res.* **2012**, *23*, 301–308.

39. Moslehi, G.; Khorasanian, D. Optimizing blocking flow shop scheduling total completion time criterion. *Comput. Oper. Res.* **2013**, *40*, 1874–1883. [CrossRef]
40. Ribas, I.; Companys, R. Efficient heuristic algorithms for the blocking flow shop scheduling problem with total flow time minimization. *Comput. Ind. Eng.* **2015**, *87*, 30–39. [CrossRef]
41. Ribas, I.; Companys, R. Xavier Tort-Martorell, An efficient Discrete Artificial Bee Colony algorithm for the blocking flow shop problem with total flowtime minimization. *Expert Syst. Appl.* **2015**, *42*, 6155–6167. [CrossRef]
42. Lia, X.; Wang, Q.; Wu, C. Efficient composite heuristics for total flowtime minimization in permutation flowshops. *Omega* **2009**, *37*, 155–164. [CrossRef]
43. Kirlik, G.; Oguz, C. A Variable Neighborhood Search for Minimizing Total Weighted Tardiness with Sequence Dependent Setup Times on a Single Machine. *Comput. Oper. Res.* **2012**, *39*, 1506–1520. [CrossRef]
44. 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]
45. 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]
46. González, M.A.; Vela, C.R. An efficient memetic algorithm for total weighted tardiness minimization in a single machine with setups. *Appl. Soft Comput.* **2015**, *37*, 506–518. [CrossRef]
47. 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.* **2016**. [CrossRef]
48. Mladenovic, N.; Hansen, P. Variable neighborhood search. *Comput. Oper. Res.* **1997**, *24*, 1097–1100. [CrossRef]
49. Tasgetiren, M.F.; Liang, Y.-C.; Sevkli, M.; Gencyilmaz, G. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *Eur. J. Oper. Res.* **2007**, *177*, 1930–1947. [CrossRef]
50. Pan, Q.-K.; Tasgetiren, M.F.; Liang, Y.-C. A Discrete Particle Swarm Optimization Algorithm for the No-Wait Flowshop Scheduling Problem with Makespan and Total Flowtime Criteria. *Comput. Oper. Res.* **2008**, *35*, 2807–2839. [CrossRef]
51. Pan, Q.-K.; Wang, L.; Tasgetiren, M.F.; Zhao, B.-H. A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion. *Int. J. Adv. Manuf. Technol.* **2008**, *38*, 337–347. [CrossRef]
52. Tasgetiren, M.F.; Liang, Y.-C.; Sevkli, M.; Gencyilmaz, G. Particle swarm optimization and differential evolution for single machine total weighted tardiness problem. *Int. J. Prod. Res.* **2006**, *44*, 4737–4754. [CrossRef]
53. Tasgetiren, M.F.; Sevkli, M.; Liang, Y.-C.; Yenisey, M.M. A particle swarm optimization and differential evolution algorithms for job shop scheduling problem. *Int. J. Oper. Res.* **2006**, *3*, 120–135.
54. 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]
55. 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]
56. Tasgetiren, M.F.; Pan, Q.K.; Liang, Y.C. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. *Comput. Oper. Res.* **2009**, *36*, 1900–1915. [CrossRef]
57. Tasgetiren, M.F.; Pan, Q.K.; Suganthan, P.N.; Chen, A.H.L. A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. *Inf. Sci.* **2011**, *181*, 3459–3475. [CrossRef]
58. Tasgetiren, M.F.; Pan, Q.K.; Suganthan, P.N.; Oner, A. A Discrete Artificial Bee Colony Algorithm for the No-Idle Permutation Flowshop Scheduling Problem with the Total Tardiness Criterion. *Appl. Math. Model.* **2013**, *37*, 6758–6779. [CrossRef]
59. 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]
60. Osman, I.; Potts, C. Simulated annealing for permutation flow-shop scheduling. *Omega* **1989**, *17*, 551–557. [CrossRef]

61. Montgomery, D.C. *Design and Analysis of Experiments*; John Wiley & Sons: Hoboken, NJ, USA, 2008.
62. 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]
63. Schenker, N.; Gentleman, J.F. On judging the significance of differences by examining the overlap between confidence intervals. *Am. Stat.* **2001**, *55*, 182–186. [CrossRef]