Computers and Operations Research 111 (2019) 177–196

Contents lists available at ScienceDirect

Computers and Operations Research

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

Metaheuristic algorithms for the hybrid flowshop scheduling problem

Hande Öztop[a], M. Fatih Tasgetiren[b,c,*], Deniz Türsel Eliiyi[d], Quan-Ke Pan[e]

[a] Department of Industrial Engineering, Yasar University, Bornova 35100, Izmir, Turkey
[b] Industrial & Systems Engineering Department, Istinye University, Zeytinburnu 34010, Istanbul, Turkey
[c] State Key Laboratory, Huazhong University of Science and Technology, Wuhan 430074, P.R. China
[d] Department of Industrial Engineering, Izmir Bakircay University, Menemen 35665, Izmir, Turkey
[e] School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200072, P.R. China

ARTICLE INFO

Article history:
Received 12 June 2018
Revised 17 April 2019
Accepted 21 June 2019
Available online 22 June 2019

Keywords:
Hybrid flowshop scheduling
Block insertion heuristic
Iterated greedy algorithm
Makespan
Total flow time

ABSTRACT

The hybrid flowshop scheduling problem (HFSP) has been widely studied in the literature, as it has many real-life applications in industry. Even though many solution approaches have been presented for the HFSP with makespan criterion, studies on HFSP with total flow time minimization have been rather limited. This study presents a mathematical model, four variants of iterated greedy algorithms and a variable block insertion heuristic for the HFSP with total flow time minimization. Based on the well-known NEH heuristic, an efficient constructive heuristic is also proposed, and compared with NEH. A detailed design of experiment is carried out to calibrate the parameters of the proposed algorithms. The HFSP benchmark suite is used for evaluating the performance of the proposed methods. As there are only 10 large instances in the current literature, further 30 large instances are proposed as new benchmarks. The developed model is solved for all instances on CPLEX under a time limit, and the performances of the proposed algorithms are assessed through comparisons with the results from CPLEX and the two best-performing algorithms in literature. Computational results show that the proposed algorithms are very effective in terms of solution time and quality. Additionally, the proposed algorithms are tested on large instances for the makespan criterion, which reveal that they also perform superbly for the makespan objective. Especially for instances with 30 jobs, the proposed algorithms are able to find the current incumbent makespan values reported in literature, and provide three new best solutions.

1. Introduction

The flowshop scheduling problem (FSP) has been widely handled in literature since the study in 1954 (Johnson, 1954). In the standard FSP, a set of n jobs must be processed on $m\,(m > 1)$ stages following the same production route, and there is single machine in each stage. To increase the efficiency of the shop floor, the standard flowshop layout is extended to hybrid flowshop, where there are parallel machines in some of the stages. This layout is also called as flexible flowshop, multi-processor flowshop or flowshop with parallel machines in literature. In the hybrid flowshop layout, each stage has at least one machine in parallel and there is strictly more than one machine in at least one of the stages. The hybrid FSP is harder to solve than the standard FSP, as it considers both assignment and scheduling of the jobs in each stage.

The hybrid flowshop layout is commonly preferred in real-life practice, as it increases and/or balances the total production capacity by reducing the effect of bottleneck stages (Ribas et al., 2010). It can be found in various sectors such as steel (Pan et al., 2013), paper (Sherali et al., 1990), textile (Grabowski and Pempera, 2000) and electronics (Wittrock, 1988; Liu and Chang, 2000; Jin et al., 2002). Due to its real-life applicability, many exact and heuristic solution approaches have been proposed for the hybrid flowshop scheduling problem (HFSP). But, most of these studies consider makespan (maximum completion time) as an objective, as this criterion is very important for increasing the utilization of resources and obtaining a high throughput. However, the total flow time objective is also critical as it effects the total capacity utilization. Since the total flow time calculates the total time that jobs spent in the system, the total in-process time and in-process inventory can be reduced by minimizing the total flow time. Furthermore, the total flow time minimization leads to a quick turn-around of the jobs and a more steady utilization of machines (Framinan and Leisten, 2003). Therefore, this criterion is very important for the companies where each job is required once it is completed and/or there are high inventory/holding costs. In the current literature, there are only few studies on the HFSP with total/average flow time minimization. Our study aims to fill this research gap

* Corresponding author at: Industrial & Systems Engineering Department, Istinye University, Zeytinburnu 34010, Istanbul, Turkey.
E-mail address: fatih.tasgetiren@istinye.edu.tr (M. Fatih Tasgetiren).

https://doi.org/10.1016/j.cor.2019.06.009
0305-0548/

by presenting new algorithms and benchmark results for this problem.

In this study, four variants of Iterated Greedy (IG) algorithm and a Variable Block Insertion Heuristic (VBIH) are proposed for the HFSP with total flow time minimization, as well as a mathematical model. Furthermore, a detailed design of experiment (DOE) is carried out to determine the best parameter values for the proposed algorithms. Through a computational study, the proposed solution approaches are compared with the two best-performing algorithms in literature, using the well-known HFSP benchmark suite (Carlier and Neron, 2000; Liao et al., 2012). Consequently, the best-known results are updated for these benchmarks. As there are only 10 large instances (Liao et al., 2012) in the current literature, additional 30 large instances are generated and proposed as new benchmarks. The makespan and total flow time results are reported for these large instances.

The remainder of this paper is organized as follows. In Section 2, a comprehensive literature review is presented and the motivation is stated. Section 3 formally defines the problem and presents the mixed integer programming formulation. Section 4 explains the proposed metaheuristic algorithms. In Section 5, a design of experiment is presented for parameter tuning. Section 6 provides the computational results to evaluate the performance of the proposed solution methods. In Section 7, the concluding remarks are presented as well as possible future research directions.

2. Literature review

The HFSP has been studied in existing literature with various objectives, machine characteristics and job constraints. Many solution approaches including exact and heuristic methods have been presented for the HFSP due to its computational complexity and real-life practicability. Two state-of-the-art reviews on HFSP are provided by Ribas et al. (2010) and Ruiz and Vazquez Rodriguez (2010), which point out that most of the existing studies consider the maximum completion time (makespan) as the sole performance measure. Other performance measures such as total/average flow time, completion time and tardiness, both in weighted and unweighted forms, are studied considerably less (Ruiz and Vazquez Rodriguez, 2010).

As exact solution methods, several branch-and-bound (B&B) algorithms were proposed to solve the HFSP with makespan criterion (Carlier and Neron, 2000; Brah and Hunsucker, 1991; Neron et al., 2001; Rajendran and Chaudhuri, 1992a; Moursli and Pochet, 2000; Portmann et al., 1998; Morita and Shio, 2005). In some studies (Portmann et al., 1998; Morita and Shio, 2005) the heuristic methods are incorporated with B&B to generate upper bound values. There are relatively fewer studies that apply exact solution methods to solve the HFSP with total flow time or completion time objectives (Rajendran and Chaudhuri, 1992b; Vignier et al., 1996; Azizoğlu et al., 2001; Tang et al., 2006). In Rajendran and Chaudhuri (1992b), the authors presented a B&B algorithm to obtain a permutation schedule minimizing total flow time. Another B&B algorithm and several lower bounds were proposed for the HFSP with total completion time minimization in Vignier et al. (1996), whereas a B&B algorithm, lower bounds and dominance rules were presented for the HFSP with total flow time minimization in Azizoğlu et al. (2001). Recently, an integer programming formulation and a Lagrangian relaxation were proposed for the HFSP with the objective of minimizing the total weighted completion time (Tang et al., 2006). A review on exact solution methods for the HFSP can be found in Kis and Pesch (2005).

Due to the NP-hard nature of the problem, exact solution methods such as B&B and mixed-integer programming can optimally solve only small-sized instances that have simple settings (Carlier and Neron, 2000; Brah and Hunsucker, 1991; Neron et al., 2001). Therefore, heuristic and metaheuristic approaches were called on for solving more complex and larger instances. In an early study, a simple heuristic was proposed for the two-stage HFSP with makespan criterion (Gupta, 1988). A simulated annealing algorithm and a tabu search algorithm were also proposed for the same problem (Haouari and M'Hallah, 1997). Various metaheuristics were also developed for the HFSP with makespan objective, such as tabu search (Haouari and M'Hallah, 1997; Nowicki and Smutnick, 1998; Negenman, 2001), ant colony optimization (ACO) (Alaykyran et al., 2007), genetic algorithm (Kahraman et al., 2008; Xiao et al., 2000), simulated annealing (SA) (Haouari and M'Hallah, 1997; Negenman, 2001; Jin et al., 2006), particle swarm optimisation (PSO) (Liao et al., 2012), artificial immune approach (AI) (Engin and Doyen, 2004), quantum-inspired immune algorithm (QIA) (Niu et al., 2009), discrete artificial bee colony (DABC) algorithm (Pan et al., 2014), improved discrete artificial bee colony (IDABC) (Cui and Gu, 2015) and IG algorithm (Kizilay et al., 2014). In Nowicki and Smutnick, (1998), Alaykyran et al. (2007), Kahraman et al. (2008), Engin and Doyen (2004), the proposed algorithms were evaluated using the well-known HFSP benchmarks (Carlier and Neron, 2000) under a 1600-second time limit, and the results reveal that all of the proposed metaheuristics solved more problems than the exact B&B method of Carlier and Neron (2000). Later, several of these algorithms were compared (Liao et al., 2012), and the experimental results showed that the PSO algorithm of Liao et al. (2012) was able to solve more benchmark problems (Carlier and Neron, 2000) with a smaller deviation than the AI (Engin and Doyen, 2004), ACO (Alaykyran et al., 2007), QIA (Niu et al., 2009), and B&B (Carlier and Neron, 2000) in 1600 s. As the benchmarks of (Carlier and Neron, 2000) were originally developed for exact solution purposes, the largest instance includes only 15 jobs and 10 stages. Therefore, Liao et al. (2012) generated 10 larger instances with 30 jobs and 5 stages, and compared their algorithm with AI (Engin and Doyen, 2004) on these instances. The results showed that PSO performed considerably better than AI (Engin and Doyen, 2004). In recent studies, the experimental results revealed that the DABC algorithm in Pan et al. (2014) and the IDABC algorithm in Cui and Gu (2015) produced significantly better results than PSO in Liao et al. (2012).

Relatively less papers proposed heuristic solution approaches for the HFSP with total/average flow time or completion time objectives. Several heuristics were proposed for the makespan and mean flow time objectives in HFSP (Brah and Loo, 1999). Afterwards, a three-phase heuristic and an SA approach were presented for the HFSP with average completion time minimization (Allahverdi and Al-Anzi, 2006). A quantum-inspired immune algorithm was also presented for the HFSP with mean flow time minimization (Niu et al., 2012). Later, an improved migrating bird optimisation with several heuristics and local search procedures was proposed to minimize total flow time (Pan and Dong, 2014). More complex settings were also considered in some studies (Low, 2005; Naderi et al., 2009). For the HFSP with unrelated parallel machines, an SA algorithm was proposed considering the total flow time as an optimization criterion (Low, 2005). Independent setup times and dependent removal times were also handled in that study. Later, an improved SA method was proposed for the HFSP with sequence-dependent setup and transportation times, in order to minimize total completion time and total tardiness (Naderi et al., 2009).

Even though many exact and heuristic solution approaches have been presented for the HFSP, most of them consider makespan. There are relatively few studies for the HFSP with total flow time minimization, and to the best of our knowledge, none of the existing studies applied the IG algorithm for this problem except (Öztop et al., 2018). In Öztop et al. (2018), two variants of IG have

been developed and the results were reported for the HFSP benchmark set (Carlier and Neron, 2000; Liao et al., 2012) with total flow time criterion for the first time in literature. However, very short runs are performed in that study (Öztop et al., 2018), suggesting a room for improvement.

In this study, we propose a traditional IG algorithm (Ruiz and Stützle, 2007), three novel IG algorithms and a novel VBIH for the problem, as well as a comprehensive design of experiment for the parameter tuning. The constructive heuristic and the local search methods of the proposed IG algorithms are different than the ones in Öztop et al. (2018). To the best of our knowledge, VBIH is not employed for the HFSP with total flow time criterion in the current literature. In this study, we compare the performances of five algorithms with two best-performing algorithms from the literature and the time-limited CPLEX results, and improve most results reported in Öztop et al. (2018). Additionally, we report optimal results for 26 instances of the well-known HFSP benchmark suite (Carlier and Neron, 2000). There are only 10 large instances (Liao et al., 2012) in the current literature on HFSP, which are not sufficient to compare performances of metaheuristics. Therefore, we generate 30 new large instances using the same parameters in Liao et al. (2012), and propose these as new benchmark problems. We also report the results of the makespan and total flow time criteria for these new instances.

3. Problem formulation

The HFSP can be seen as a combination of two scheduling problems: the standard flowshop and the parallel shop. In HFSP, there are m stages in series, where $m > 1$. A set of n jobs must be sequentially processed in these stages following the same production flow: Stage 1, stage 2, ..., stage m. Each job $j \in J$ has an uninterrupted and non-negative processing time p_{kj} in stage $k \in M$. There are $|I_k| \geq 1$ identical parallel machines in each stage $k \in M$, where $|I_k| > 1$ in at least one stage. As parallel machines within a stage are identical, a job can be processed by any machine $i \in I_k$ at stage k. At a time, a machine can process only one job and a job can be processed by only one machine. All machines and jobs are available to be processed at time zero and job preemption is not allowed. The capacity of intermediate buffers between stages is unlimited. The travel time between consecutive stages and the setup times are included in the processing times of jobs at the corresponding stage. The problem data is assumed to be deterministic and known in advance. The notation for the problem is provided in Table 1.

The HFSP aims to find a schedule that optimizes a given objective. In this study, we consider two common objectives: minimizing the total flow time (total completion time) and minimizing the maximum completion time (makespan). The former problem is denoted as FHm, $((PM^{(k)})_{k=1}^{m})||\bar{F}$ and the latter is denoted as FHm, $((PM^{(k)})_{k=1}^{m})||C_{max}$ according to the notation proposed by Vignier et al. (1999), which follows the three field notation of Graham et al. (1979). The problem is NP-hard since the HFSP with only two stages, which has only one machine in one of the stages, is known to be NP-hard (Gupta, 1988).

Table 1
Problem notation.

Sets	
M	Set of stages {1,2,...,m}
J	Set of jobs
I_k	Set of machines at stage $k \in M$
Parameters	
p_{kj}	Processing time of job $j \in J$ at stage $k \in M$
Q	A very large number
Decision variables	
s_{kj}	Starting time of job j at stage k
x_{jik}	1 if job j is processed at machine i at stage k, 0 otherwise
y_{kjr}	1 if job j precedes job r at stage k, 0 otherwise
C_{max}	Maximum completion time (makespan)
C_j	Completion time of job j

The mixed integer programming (MIP) model of the HFSP with total flow time minimization objective, which is similar to the one in Pan and Dong (2014) is given as follows.

$$Minimize \sum_{j \in J} C_j \tag{1}$$

subject to

$$s_{mj} + p_{mj} = C_j \qquad \forall j \in J \tag{2}$$

$$\sum_{i \in I_k} x_{jik} = 1 \qquad \forall j \in J,\ k \in M \tag{3}$$

$$s_{kj} + p_{kj} \leq s_{k+1,j} \qquad \forall j \in J,\ (k, k+1) \in M \tag{4}$$

$$s_{kj} - (s_{kr} + p_{kr}) + Q\left(2 + y_{kjr} - x_{jik} - x_{rik}\right) \geq 0$$
$$\forall j, r \in J : j < r, k \in M,\ i \in I_k \tag{5}$$

$$s_{kr} - \left(s_{kj} + p_{kj}\right) + Q\left(3 - y_{kjr} - x_{jik} - x_{rik}\right) \geq 0$$
$$\forall j, r \in J : j < r, k \in M, i \in I_k \tag{6}$$

$$s_{kj} \geq 0 \quad \forall k \in M, j \in J \quad y_{kjr} \in \{0,\ 1\} \quad \forall j, r \in J,\ k \in M$$
$$x_{jik} \in \{0,\ 1\} \quad \forall k \in M,\ j \in J,\ i \in I_k \tag{7}$$

The objective function (1) minimizes the total flow time. Constraint set (2) calculates the flow time (completion time) of each job. Constraint set (3) guarantees that each job passes through all stages and is processed by exactly one machine at each stage. Constraint set (4) ensures that the next operation of a job starts only after its previous operation has been completed. Constraint sets (5) and (6) prevent the overlapping of any two jobs on the same machine and define the sequence of the jobs, where $Q \geq \sum_{j \in J} \sum_{k \in M} |I_k| p_{kj}$. For any two jobs assigned to same machine, the next job can start after the previous one is completed. Constraint set (7) defines the domains of decision variables.

The MIP model of the HFSP with makespan objective is given below. This model is identical to the above except that the objective function (1) is changed to minimize makespan (8), and constraint set (2) is replaced by constraint set (9) calculating the maximum completion time of jobs in the last stage.

$$Minimize\ C_{\max} \tag{8}$$

subject to
(3) – (7) and

$$s_{mj} + p_{mj} \leq C_{max} \ \forall j \in J \tag{9}$$

4. Metaheuristic algorithms

We propose five metaheuristic algorithms for the problem as four variants of IG and VBIH. In all algorithms, we use forward scheduling for fitness value calculation, which is very common in the HFSP literature. In forward scheduling, the jobs are assigned to the most available machine in the first stage with the order of initial solution π. Afterwards, in each following stage, the jobs are ordered with regard to their release times from the previous stage and assigned to the most available machine in that order. Note that the order of jobs can be different in each stage. In this manner, a complete schedule can be found by using only an initial solution π.

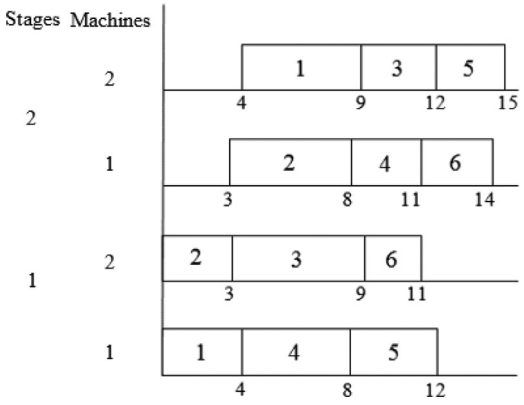

Fig. 1. Gantt chart with TFT = 69 and C_{max} = 15.

The forward schedule approach is demonstrated with an instance that has 6 jobs and 2 stages. Given the processing times as $p_{kj} = \begin{pmatrix} 4 & 3 & 6 & 4 & 4 & 2 \\ 5 & 5 & 3 & 3 & 3 & 3 \end{pmatrix}$ and an initial solution as $\pi = \{1, 2, 3, 4, 5, 6\}$, the jobs are scheduled in a forward manner as follows. At the first stage, the jobs are assigned to the most available machine in the permutation of the initial solution. Next, the jobs are re-ordered with respect to their release times from the first stage so the resulting job order becomes {2, 1, 4, 3, 6, 5}. The second stage assignments are made following this new ordering. The whole Gantt chart for the initial solution π is illustrated in Fig. 1 with a total flow time (TFT) of 69 and a makespan (C_{max}) of 15.

4.1. Constructive heuristics

We use two types of constructive heuristics to generate the initial solutions, namely the NEH heuristic (Nawaz et al., 1983) and the GRASP_NEH(x). The pseudo-code of the well-known NEH heuristic is given in Fig. 2. In the first phase, the sum of the processing times on all stages (TP_j) is calculated for each job $j \in J$. Then, jobs are sorted in decreasing order of TP_j to obtain $\delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$. In the second phase, the first job in δ (δ_1) is selected to establish a partial solution of length one. Then, the remaining jobs in δ are sequentially inserted into the partial solution one by one until a complete solution with n jobs is obtained. Namely, in the i^{th} iteration, the job δ_i is chosen and tentatively inserted into all the i possible positions of the partial solution π. Then, job δ_i is inserted into the best position in partial solution π that results in the minimum fitness value. Once all jobs in δ have been inserted, a complete solution is obtained. In the NEH heuristic, there are $(n(n+1)/2) - 1$ insertions in total and each insertion generates a schedule. As all schedules must be evaluated, the computational complexity is $O(n^3m)$ (Pan et al., 2014). From here on, in all figures throughout the paper, $f(\pi)$ refers to the fitness value of a solution π, namely the total flow time (TFT) or the makespan (C_{max}) value.

In this paper, we propose a GRASP_NEH(x) heuristic. The Greedy Randomized Adaptive Search Procedure (GRASP) is a stochastic constructive heuristic developed by Feo and Resende (1995). In the GRASP, initial solutions are created by a greedy randomized procedure. In this procedure, from a set of candidate jobs to be sequenced, the next job to be appended is selected using a cost function. The cost of each candidate is computed and a restricted candidate list (RCL) is constructed, which holds jobs with a cost no larger than $cf_{min} + \alpha\ (cf_{max} - cf_{min})$, where cf_{min} and cf_{max} are the minimum and maximum costs. The threshold value $\alpha \in [0, 1]$ limits the size of RCL and regulates balance between greedy and randomized procedures. $\alpha = 0$ states a pure greedy approach, i.e. selecting the next job with a minimum cost, while $\alpha = 1$ indicates a randomized selection. Next, a job is randomly selected from RCL and appended to the next position in the sequence. The candidate job list is updated and the costs are re-computed. The procedure continues until the set of candidate jobs is empty. In this study, we combine the greedy randomized procedure with the NEH heuristic to further improve the solutions.

The overall GRASP_NEH(x) procedure is outlined in Fig. 3. Initially, we sort the jobs in a decreasing order of their total processing times to define the initial order of jobs, as in NEH. From this initial order of jobs (π^*), we generate x new solutions by sending a different job position index h to GRASP(π^*, h) procedure, where $h = 1 \ldots x$. Namely, in the h^{th} iteration, GRASP(π^*, h) generates a solution employing a greedy randomized procedure, as given in Fig. 4. Then, the second phase of NEH (insertion procedure) is applied to the complete solution obtained from GRASP(π^*, h). If the resulting solution π'' after insertion procedure is better than the solution π' obtained from GRASP(π^*, h), π'' is accepted as π^h. Otherwise, the solution π' obtained from GRASP(π^*, h) is recorded as π^h.

Note that, the job position index h is used to define the first job in the GRASP(π^*, h) procedure. As the first job has an impact on the solution quality, the first job of the solution should be chosen carefully. Initially, the first job of the initial order (π_1^*) is taken as the first job, and the GRASP(π^*, h) and insertion procedures are applied respectively to generate a new solution (π^1). Next, the second job of the initial order (π_2^*) is taken as the first job, and the same procedures are applied to generate another new solution (π^2). This process is repeated x times for generating x new solutions. Finally, the best one among these x solutions is chosen as the final solution. We fix the number of solutions x as n in this study. As GRASP_NEH(x) applies these procedures n times to generate n solutions, its computational complexity becomes $O(n^4m)$.

The GRASP (π^*, h) procedure is given in Fig. 4. Suppose that $i-1$ jobs are scheduled with a partial solution $\{\pi_1, \pi_2, .., \pi_{i-1}\}$. Job π_i is the next job to be appended into the partial solution, which can be any job from the set of unscheduled jobs U. A problem-specific cost function is necessary to determine job π_i, which is selected as the objective function in this study. For the HFSP with TFT criterion, we compute the total flow time TFT^j for each job $j \in U$ by appending job j to the partial solution ($\pi_i = j$), and define

NEH Heuristic

for each job j calculate the total processing time on the m stages: $\forall j \in J, TP_j = \sum_{k=1}^{m} p_{kj}$
Step1. δ = *Sort the jobs in decreasing order of* TP_j
Step2. $\pi = \{\delta_1\}$ %initialize partial solution with the first job in δ
  for $(i = 2$ *to* $n)$ *do* %loop over remaining jobs
    Take job δ_i *from* δ *and test it in all of the i possible positions of solution* π
    Insert job δ_i *to the best position in solution* π *that results in the minimum fitness value*
  end for
return π

Fig. 2. NEH Heuristic.

GRASP_NEH(x) Heuristic

for each job j calculate the total processing time on the m stages: $\forall j \in J, TP_j = \sum_{k=1}^{m} p_{kj}$

Step1. $\pi^* =$ *Sort the jobs in decreasing order of* TP_j

Step2. *for* $(h = 1$ *to* $x)$ *do*

- *Take* π_h^* *as the first job*
- $\pi' = GRASP(\pi^*, h)$ % generate a sequence by greedy randomized procedure
- $\pi'' =$ *Apply the second phase of NEH to* π'*, as follows*:
  - $\pi'' = \{\pi_1'\}$
  - *for* $(i = 2$ *to* $n)$ *do*
    - *Take job* π_i' *from* π' *and test it in all of the i possible positions of solution* π''
    - *Insert job* π_i' *to the best position in* π'' *that results in the minimum fitness value*
  - *end for*
- *if* $\left(f(\pi'') < f(\pi')\right)$
  - $\pi^h = \pi''$
  - *else*
  - $\pi^h = \pi'$
  - *end if*
- *Record solution* π^h

end for

Step3. *Return the best solution amongst* $\{\pi^1, \pi^2, .., \pi^x\}$

Fig. 3. GRASP_NEH(***x***) Heuristic.

Procedure $GRASP(\pi^*, h)$

*Step*1. *Let U is the set of unscheduled jobs, initially* $U = J$

*Step*2. *Set* $\pi_1 = \pi_h^*, U = J - \{\pi_1\}$

for $(i = 2$ *to* $n)$ *do*

- *for each job* $j \in U$
  - *Calculate the objective function value when* $\pi_i = j$ *and denote as* CF_j
  - *end for*
- *Calculate* CF_{min} *and* CF_{max}
- *for each job* $j \in U$
  - *Put job j into RCL if it satisfies the condition* (10)
  - *end for*
- *Randomly select a job j from RCL and append it into the partial solution as* $\pi_i = j$
- $U = U - \{\pi_i\}$ %remove the selected job from U

end for

return π

Fig. 4. GRASP(π^*, h) procedure.

a cost function as $CF_j = TFT^j$. Similarly, for the HFSP with C_{max} criterion, we compute the makespan (C_{max}^j) for each job j and use $CF_j = C_{max}^j$ as a cost function. We calculate minimum (CF_{min}) and maximum (CF_{max}) among these values and fill jobs into RCL if any of them satisfies the following condition:

$$CF_j - CF_{min} \leq \alpha * (CF_{max} - CF_{min}) \tag{10}$$

Here, α is the threshold value and after preliminary experiments, it is determined as $\alpha = 0.1$. Finally, job π_i is selected randomly from RCL as the next job to be appended to the partial solution.

4.2. Iterated greedy algorithms

The IG algorithm has mainly four components; namely, the initial solution, the destruction-construction (DC) procedure, local search, and the acceptance criterion (Ruiz and Stützle, 2007). In IG, the initial solution is constructed via a constructive heuristic and the new solutions are generated by DC, where dS jobs are randomly removed from current solution π and are re-inserted into the partial solution in a sequential manner. A local search follows, and an acceptance criterion is imposed on the solution after local search. These basic steps are repeated until a stopping criterion is satisfied. In this study, we propose four variants of IG, which are explained in the following subsections.

4.2.1. IG_{RS} and IG_{GR} algorithms

The first variant of the IG algorithm is the traditional IG_{RS}, proposed in Ruiz and Stützle (2007), in which the initial solution is constructed by NEH. In destruction, dS jobs are randomly chosen from the solution without repetition. Then, in construction, these dS jobs are re-inserted into the partial solution in the order in which they were removed following a best insertion policy, until a complete solution of n jobs is obtained.

After destruction and construction, a local search aims to further enhance solution quality. The IG_{RS} algorithm employs a first-improvement insertion local search, which is outlined in Fig. 5. For each job π_i, the procedure inserts job π_i into all possible positions of solution π. When the most improving insertion position is found, job π_i is inserted into that position. These steps are repeated for all jobs. If an improvement is found, the local search is re-run until no better solution is obtained.

After local search, if the obtained solution is at least as good as the incumbent solution, it is accepted. Otherwise, the new solution is accepted using a simulated annealing-like acceptance criterion with a constant temperature T, which is computed as follows (Osman and Potts, 1989):

$$T = \frac{\sum_{j=1}^{n}\sum_{k=1}^{m} p_{kj}}{10nm} \times \tau P, \tag{11}$$

where n is the number of jobs, m is the number of stages and τP is a parameter to be adjusted. Note that, the value of temperature T depends on the particular instance data (Osman and Potts, 1989).

First − Improvement Insertion Local Search (π)

improvement = *true*
while (*improvement* = *true*) *do*
  improvement = *false*
  for (i = 1 *to* n) *do*
    π^* = *Insert job* π_i *to the best position in solution* π, *as follows*:
      - *Remove job* π_i *from* π
      - *Test job* π_i *in all of the possible positions of* π
      - *Insert job* π_i *to the best position in* π *that results in the minimum fitness value*
    if $\left(f(\pi^*) < f(\pi)\right)$ *then do*
      $\pi = \pi^*$
      improvement = *true*
    end if
  end for
end while
return π

Fig. 5. First-improvement insertion local search.

Procedure IG_{RS} *Algorithm*($dS, \tau P$)

π^0 = *NEH* %initial solution
$\pi^{best} = \pi^0$
while (*time limit is not exceeded*) *do*
  π^1 = *Destruction* (π^0, dS) %remove dS jobs randomly from the solution π^0
  π^2 = *Construction* (π^1, dS) %re-insert dS jobs into the solution π^1 following a best insertion policy
  π^3 = *First − Improvement Insertion Local Search* (π^2) %local search to the complete solution
  if $f(\pi^3) \le f(\pi^0)$ *then*
    $\pi^0 = \pi^3$
    if $f(\pi^3) < f(\pi^{best})$ *then*
      $\pi^{best} = \pi^3$
    end if
  else if $\left(r < exp\{-\left(f(\pi^3) - f(\pi^0)\right)/T\}\right)$ *then*
    $\pi^0 = \pi^3$
  end if
end while
return π^{best} *and* $f(\pi^{best})$

Fig. 6. IG_{RS} algorithm.

The pseudo-code of the traditional IG_{RS} is provided in Fig. 6, where r is a uniform random number between 0 and 1.

As a second variant of the IG algorithm, we present an IG_{GR} algorithm. The IG_{GR} algorithm is identical to IG_{RS} except that IG_{GR} employs GRASP_NEH(x) heuristic to generate the initial solution, where $x = n$. Consequently, the pseudo-code of the IG_{GR} algorithm is same as in Fig. 6, only change is that the initial solution is constructed by GRASP_NEH(x) heuristic in IG_{GR} (first line: π^0 = GRASP_NEH(n)) instead of NEH.

4.2.2. RIS and RSS local search procedures

For the rest of the algorithms in this paper, we employ two very effective local search algorithms based on swap and insertion neighborhoods. The first one is the Referenced Insertion Scheme (RIS) (Tasgetiren et al., 2017; Pan et al., 2008), which is guided by a reference solution π^R, namely, the best solution obtained so far during the search procedure. The RIS local search guides the job removal by a good reference solution instead of a random selection. It assumes that positions of the jobs in the current solution are not appropriate according to the reference solution, thus, it leads jobs to find better positions to be inserted in the current solution. For instance, if the reference solution is given by $\pi^R = \{2, 4, 1, 3, 5\}$ and the current solution by $\pi = \{1, 2, 3, 4, 5\}$, the RIS removes job 2 from the current solution and inserts it into all possible positions of π except its current position: {**2**, 1, 3, 4, 5}, {1, 3, **2**, 4, 5}, {1, 3, 4, **2**, 5}, {1, 3, 4, 5, **2**}. A new solution with the best insertion is replaced with the current solution and the iteration counter is reset if any improvement occurs. Otherwise, the counter is incremented by one. Next, the procedure removes job 4 from the current solution and inserts it into all possible positions except its current position. The procedure is repeated until the iteration counter is greater than n, and a new solution is obtained. The outline of RIS is presented in Fig. 7.

The second local search algorithm is based on swap neighborhood structure. We employ a referenced swap local search, denoted as RSS in literature (Tasgetiren et al., 2016a). Similar to RIS, the reference solution guides the search process. The referenced job is first found in the current solution and swapped with all other jobs in the current solution. Regarding the above example with $\pi^R = \{2, 4, 1, 3, 5\}$ and $\pi = \{1, 2, 3, 4, 5\}$, the RSS swaps job 2 with all other jobs in π: {**2**, **1**, 3, 4, 5}, {1, **3**, **2**, 4, 5}, {1, **4**, 3, **2**, 5}, {1, **5**, 3, 4, **2**}. The best move is retained. The iteration counter is reset in case of improvement, and incremented by one otherwise. The procedure repeats until the iteration counter is greater than n. Note that, the RSS is very similar to RIS, except that RSS employs swap neighborhood structure instead of insertion neighborhood structure. For the sake of completeness, the outline of RSS local search is provided in Fig. 8.

4.2.3. IGT and IGT_{ALL} algorithms

As a third variant of the IG algorithm, we propose a new algorithm in this study as IGT. Like the IG_{GR} algorithm, the initial solution is generated by GRASP_NEH(x) in IGT. Next, the destruction-construction (DC) is applied as in IG_{RS}. After DC, the

Procedure RIS Local Search (π, π^{best})

$\pi^R = \pi^{best}$, $Iter = 1$, $pos = 1$
while $(Iter \leq n)$ *do*
- $k = 1$
- *Find the position of job* π^R_{pos} *in the solution* π *, as follows*:
  while $(\pi_k \neq \pi^R_{pos})$
  $k = k + 1$
  end while
- $pos = pos + 1$
- *if* $(pos = n + 1)$ *then*
  $pos = 1$
  end if
- *Remove job* π_k *from* π
- $\pi^* =$ *Insert job* π_k *into the best position in* π %insertion neighborhood structure
- *if* $\left(f(\pi^*) < f(\pi)\right)$ *then do*
  $\pi = \pi^*$
  $Iter = 1$
  else
  $Iter = Iter + 1$
  end if

end while
return π

Fig. 7. RIS local search.

Procedure RSS Local Search (π, π^{best})

$\pi^R = \pi^{best}$, $Iter = 1$, $pos = 1$
while $(Iter \leq n)$ *do*
- $k = 1$
- *Find the position of job* π^R_{pos} *in the solution* π *, as follows*:
  while $(\pi_k \neq \pi^R_{pos})$
  $k = k + 1$
  end while
- $pos = pos + 1$
- *if* $(pos = n + 1)$ *then*
  $pos = 1$
  end if
- $\pi^* =$ *Swap job* π_k *with all other jobs in* π *and keep the best move* %swap neighborhood structure
- *if* $\left(f(\pi^*) < f(\pi)\right)$ *then do*
  $\pi = \pi^*$
  $Iter = 1$
  else
  $Iter = Iter + 1$
  end if

end while
return π

Fig. 8. RSS local search.

algorithm employs RIS and RSS local search algorithms with a jumping probability *jP*. The solution is accepted according to the same rules in IG_{RS}. In the outline of the IGT algorithm given in Fig. 9, r is a uniform random number between 0 and 1.

Recently, a new algorithm is presented in the literature as IG_{ALL} (Dubois-Lacoste et al., 2017) with excellent results for the permutation flowshop 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 the solution quality (Dubois-Lacoste et al., 2017). Inspiring from this algorithm, we propose IGT_{ALL} as a fourth variant of IG. The IGT_{ALL} algorithm is identical to IGT except that IGT_{ALL} applies the first-improvement insertion local search in Fig. 5 to the partial solutions after destruction. The initial solution is constructed with GRASP_NEH(x), and both RIS and RSS are applied to the complete solution with a jumping probability *jP*. The solution is accepted according to the same procedures in IG_{RS} and IGT.

The following example can clarify the difference between IGT_{ALL} and IGT. Suppose that the current solution is $\pi = \{2, 1, 4, 3, 5\}$. IGT first destructs the solution by randomly removing some jobs from the current solution π. Let the destruction size be $dS = 2$. It means two jobs are removed randomly from the current solution, say jobs 1 and 5. The two partial solutions obtained after destruction are therefore $\pi^d = \{1, 5\}$ and $\pi^c = \{2, 4, 3\}$. Now, the construction procedure inserts job 1 into four positions as follows: $\pi^{c1} = \{\mathbf{1}, 2, 4, 3\}$, $\pi^{c2} = \{2, \mathbf{1}, 4, 3\}$, $\pi^{c3} = \{2, 4, \mathbf{1}, 3\}$, and

Procedure IGT Algorithm $(dS, \tau P, jP)$

π^0 = GRASP_NEH(n) %initial solution
$\pi^{best} = \pi^0$
while (*time limit is not exceeded*) *do*
  $\pi^1 = Destruction\ (\pi^0, dS)$ %remove dS jobs randomly from the solution π^0
  $\pi^2 = Construction\ (\pi^1, dS)$ %re-insert dS jobs into the solution π^1 following a best insertion policy
  if $(r < jP)$
    $\pi^3 = RIS\ Local\ Search\ (\pi^2, \pi^{best})$ %local search to the complete solution
  else
    $\pi^3 = RSS\ Local\ Search\ (\pi^2, \pi^{best})$ %local search to the complete solution
  end if
  if $f(\pi^3) \le f(\pi^0)$ *then do*
    $\pi^0 = \pi^3$
    if $f(\pi^3) < f(\pi^{best})$ *then do*
      $\pi^{best} = \pi^3$
    end if
  else if $\left(r < exp\{-(f(\pi^3) - f(\pi^0))/T\}\right)$
    $\pi^0 = \pi^3$
  end if
end while
return π^{best} *and* $f(\pi^{best})$

Fig. 9. IGT algorithm.

$\pi^{c4} = \{2, 4, 3, \mathbf{1}\}$. Among these four partial solutions, if the best fitness value belongs to partial solution $\pi^{c3} = \{2, 4, 1, 3\}$ for instance, the construction procedure proceeds from this solution and inserts job 5 into five positions as follows: $\pi^{c1} = \{\mathbf{5}, 2, 4, 1, 3\}$, $\pi^{c2} = \{2, \mathbf{5}, 4, 1, 3\}$, $\pi^{c3} = \{2, 4, \mathbf{5}, 1, 3\}$, $\pi^{c4} = \{2, 4, 1, \mathbf{5}, 3\}$, and $\pi^{c5} = \{2, 4, 1, 3, \mathbf{5}\}$. The construction procedure determines the best one among these five solutions as a complete solution. Suppose that the final solution is $\pi = \pi^{c2} = \{2, 5, 4, 1, 3\}$.

Different from the above procedure, IGT_{ALL} applies the first-improvement insertion local search in Fig. 5 to the partial solution $\pi^c = \{2, 4, 3\}$ before construction begins. Namely, the insertion local search removes job 2 from π^c and inserts it into all possible positions. The most improving insertion is chosen and compared to $f(\pi^c)$. If the new partial solution is better, it replaces the partial solution π^c. This is repeated for all jobs in π^c and the best partial solution is found. Let the best partial solution be $\pi^c = \{4, 2, 3\}$ for the above example. The construction procedure then re-inserts each job in π^d into this partial solution and determines the best complete solution. Note that this new completion of the partial solution can be different from the one found by IGT, for example $\pi = \{4, 1, 5, 2, 3\}$ can be obtained by IGT_{ALL}. Subsequently, a local search with jumping probability jP and an acceptance criterion are applied as shown in Fig 10, where r is a uniform random number between 0 and 1.

4.3. Variable block insertion heuristic

Insertion or swap neighborhood structures are commonly used in local search. Recently, block move-based local search algorithms are proposed for single machine-scheduling problems (Subramanian et al., 2014; Xu et al., 2014) where the main idea is to employ larger neighborhood move structures. In Subramanian et al. (2014), various neighborhoods are proposed such as swap, insertion, 2-block insertion, 3-block insertion and block reverse. A block move structure is proposed in Xu et al. (2014), which corresponds to removing a block with b consecutive jobs, and inserting it into another position. In Xu et al. (2014), a block move is specified by a triplet (i, k, b), where i represents the position of the first job of the block, k denotes the target insertion position of the block, and b indicates the block size. Later studies proposed Variable Block Insertion Heuristic (VBIH) algorithms (Tasgetiren et al., 2016a,b,2017). Unlike the standard block moves, the block size b changes in VBIH, allowing for several block moves with different block sizes. In this study, we propose a similar VBIH algorithm, which is outlined in Fig. 11. Note that tP is temperature parameter for the acceptance criterion and r is a uniform random number between 0 and 1.

As shown in Fig. 11, the VBIH algorithm has a minimum block size (b_{min}), and a maximum block size (b_{max}). The algorithm removes a block of jobs with size b $(\pi^{(b)})$ from the current solution. Similar to IGT_{ALL}, it applies the first-improvement insertion local search to the partial solution after removing the block. Next, $n - b + 1$ block insertion moves are randomly performed on the partial solution and the best solution is selected for applying local search. As in IGT_{ALL}, RIS or RSS is applied to the complete solution after block moves with a jumping probability jP. If the new solution obtained after the local search is incumbent, it replaces the current solution. The same block size b is retained as long as it is improving. Otherwise, the block size is incremented by one, and a simulated annealing-like acceptance criterion is used to accept the new solution in order to escape local minima. The process is repeated until the block size exceeds its maximum limit b_{max} .

The following example illustrates block insertion. Suppose that we are given a current solution $\pi = \{1, 2, 3, 4, 5\}$. Furthermore, assume that the block size is $b = 2$ and chosen block is $\pi^{(b)} = \{3, 4\}$. This yields a partial solution, $\pi^p = \{1, 2, 5\}$. After applying local search to the partial solution, suppose that we have a new partial solution $\pi^p = \{2, 1, 5\}$. Next, block $\pi^{(b)}$ is inserted into four possible positions as follows: $\pi^{p1} = \{\mathbf{3, 4}, 2, 1, 5\}$, $\pi^{p2} = \{2, \mathbf{3, 4}, 1, 5\}$, $\pi^{p3} = \{2, 1, \mathbf{3, 4}, 5\}$ and $\pi^{p4} = \{2, 1, 5, \mathbf{3, 4}\}$. Among these four solutions, the best should be chosen as the final solution.

5. Design of experiments for parameter calibration

In this section, in order to calibrate the parameters of the proposed heuristics, we carry out a detailed design of experiment (DOE) (Montgomery, 2008) for IGT_{ALL} and VBIH with TFT criterion. For this purpose, we generate random instances with 5 stages using the method proposed in Liao et al. (2012), where the number of machines in each stage is randomly generated between 3 and 5 and the processing times of the jobs are uniform in the range [1,100]. Problem instances with 30, 40, 50 and 60 jobs are

Procedure IGT_{ALL} *Algorithm* $(dS, \tau P, jP)$

$\pi^0 = \text{GRASP\_NEH}(n)$ %initial solution
$\pi^{best} = \pi^0$
while (*time limit is not exceeded*) *do*
  $\pi^1 = Destruction\ (\pi^0, dS)$ %remove dS jobs randomly from the solution π^0
  $\pi^2 = First - Improvement\ Insertion\ Local\ Search\ (\pi^1)$ %local search to the partial solution π^1
  $\pi^3 = Construction\ (\pi^2, dS)$ %re-insert dS jobs into the solution π^2 following a best insertion policy
  if $(r < jP)$
    $\pi^4 = RIS\ Local\ Search\ (\pi^3, \pi^{best})$ %local search to the complete solution
  else
    $\pi^4 = RSS\ Local\ Search\ (\pi^3, \pi^{best})$ %local search to the complete solution
  end if
  if $f(\pi^4) \le f(\pi^0)$ *then do*
    $\pi^0 = \pi^4$
    if $f(\pi^4) < f(\pi^{best})$ *then do*
      $\pi^{best} = \pi^4$
    end if
  else if $\left(r < exp\{-(f(\pi^4) - f(\pi^0))/T\}\right)$
    $\pi^0 = \pi^4$
  end if
end while
return π^{best} *and* $f(\pi^{best})$

Fig. 10. IGT_{ALL} algorithm.

Procedure VBIH Algorithm $(b_{max}, \tau P, jP)$

$\pi = GRASP\_NEH(n)$ %initial solution
$\pi^{best} = \pi$
$b_{min} = 2$
while (*time limit is not exceeded*)
  $b = b_{min}$
  do
    $\pi^1 = Remove\ block\ \pi^{(b)} from\ \pi$
    $\pi^2 = First - Improvement\ Insertion\ Local\ Search\ (\pi^1)$ %local search to the partial solution π^1
    $\pi^3 = Insert\ block\ \pi^{(b)}\ into\ the\ best\ position\ in\ \pi^2$
    if $(r < jP)$
      $\pi^4 = RIS\ Local\ Search\ (\pi^3, \pi^{best})$ %local search to the complete solution
    else
      $\pi^4 = RSS\ Local\ Search\ (\pi^3, \pi^{best})$ %local search to the complete solution
    end if
    if $\left(f(\pi^4) \le f(\pi)\right)$ *then do*
      $\pi = \pi^4$
      if $\left(f(\pi^4) < f(\pi^{best})\right)$ *then do*
        $\pi^{best} = \pi^4$
      end if
    else
      $b = b + 1$
      if $\left(r < exp\{-(f(\pi^4) - f(\pi))/T\}\right)$
        $\pi = \pi^4$
      end if
    end if
  until$(b > b_{max})$
end while
return π^{best} *and* $f(\pi^{best})$

Fig. 11. Variable block insertion heuristic.

generated, each size having 30 instances, summing up to 120 instances. The seed numbers used in random number generators in this section are different than those used for computational comparisons in Section 6. Both algorithms are 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.

In the DOE approach, the proposed IGT_{ALL} algorithm has three parameters; the destruction size (dS), the temperature adjustment parameter (τP), and the local search jumping probability (jP). After pilot experiments, we set dS to seven levels as $dS \in (2, 3, 4, 5, 6, 7, 8)$; τP to five levels as $\tau P \in (0.1, 0.2, 0.3, 0.4, 0.5)$; and jP to eleven levels as $jP \in (0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)$, resulting in $7 \times 5 \times 11 = 385$ treatments. There are 30 instances for each job size, and each instance is run for 385 treatments with a maximum CPU time $T_{max} = 30 \times n \times m$ milliseconds. We calculate the relative percent deviation (RPD) for each

Table 2
ANOVA results for parameters of IGT_{ALL}.

Source	*DF*	*Seq SS*	*Adj SS*	*Adj MS*	*F – Ratio*	*p – value*
dS	6	0.221439	0.221439	0.036907	16.14	0.000
tP	4	0.003009	0.003009	0.000752	0.33	0.858
jP	10	0.100192	0.100192	0.010019	4.38	0.000
*dS*tP*	24	0.067872	0.067872	0.002828	1.24	0.211
*dS*jP*	60	0.166020	0.166020	0.002767	1.21	0.161
*tP*jP*	40	0.102090	0.102090	0.002552	1.12	0.303
Error	240	0.548964	0.548964	0.002287		
Total	384	1.209587				

instance-treatment pair as follows:

$$RPD(TFT_i) = \left(\frac{TFT_i - TFT_{min}}{TFT_{min}}\right) * 100 \quad (12)$$

where TFT_i is the TFT value generated by a given heuristic in treatment i, and TFT_{min} is the minimum TFT found among 385 treatments. For each job size-treatment pair, average RPD values are obtained over 30 instances. Then, the response variable (ARPD) of each treatment is obtained by averaging these RPD values over all job sizes.

An ANOVA procedure is performed after determining the ARPD values for each treatment, and the results are given in Table 2. Before applying ANOVA, three main hypothesis should be tested for residuals: normality, homogeneity of variance and independence (Montgomery, 2008). The residuals from the experimental results were analyzed and all three hypothesis could be accepted. In ANOVA table, *DF* represents the degrees of freedom, *Seq SS* indicates the sequential sum of squares, *Adj SS* represents the adjusted sum of squares and *Adj MS* indicates the adjusted mean squares (*Adj SS*/*DF*), where sum of squares represents a measure of variation. Note that, *Seq SS* depends on the order the factors are introduced into model, while *Adj SS* does not depend on the order. If the design is orthogonal and balanced, adjusted and sequential sum of squares are same for each term. In ANOVA table, the *p*-value is an important measure for statistical significance. If the *p*-value is less than or equal to a desired significance level, it means that there is a statistically significant difference between the levels of the considered factor. The F-ratio is also important, which is the ratio between the variance explained by a factor (*Adj MS*) and the unexplained variance ($Adj\ MS_{Error}$). The greater F-ratio shows that the considered factor has more effect over the response variable.

The ANOVA results indicate that the destruction size and the local search jumping probability are very significant as their *p*-values are zero. In particular, *dS* is the most significant factor as it has the greatest *F*-ratio. On the other hand, different levels for the temperature adjustment parameter do not result in statistically significant differences in the ARPD value, as the *p*-value of τP is greater than the significance level, $\alpha = 0.05$. This indicates that our IGT_{ALL} performs rather robustly with respect to τP. Furthermore, no statistically significant interaction effect exists between parameters as *p*-values of parameter interaction effects are higher than the significance level. As there is no interaction, the main effects plots of the parameters can be used to determine the best parameter values for IGT_{ALL}. These plots are provided in Fig. 12.

As mentioned above, the most significant factor affecting performance is *dS*. As shown in Fig. 12, high *dS* levels result in worse ARPD values and decreased algorithm performance. Therefore, we set $dS = 2$, which corresponds to the minimum ARPD value. A similar pattern for *dS* was also observed in the original IG_{ALL} for the permutation flowshop problem (Dubois-Lacoste et al., 2017).

The *jP* parameter has the second largest *F*-ratio in Table 2. As shown in Fig. 12, a setting of $jP = 0.4$ has a better ARPD value than others. Hence, *jP* is taken as 0.4. Note that, $jP = 0$ entails that only RSS is applied while $jP = 1$ points to a pure RIS application. Using a setting of $jP = 0.4$ in IGT_{ALL} suggests that employing both local searches with a probability of 0.4 is more effective in terms of ARPD. The plots demonstrate no statistically significant difference between various levels of τP. Yet, a setting of $\tau P = 0.5$ provides a slightly better ARPD value than others. Consequently, we set the parameters of IGT_{ALL} as $dS = 2$, $\tau P = 0.5$ and $jP = 0.4$.

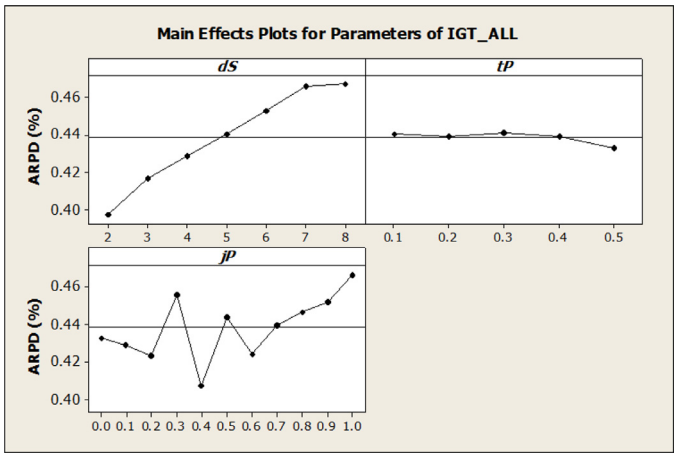

Fig. 12. Main effects plot for parameters of IGT_{ALL}.

Table 3
ANOVA results for parameters of VBIH.

Source	*DF*	*Seq SS*	*Adj SS*	*Adj MS*	*F – Ratio*	*p – value*
bMax	6	0.006568	0.006568	0.001095	1.24	0.288
tP	4	0.008779	0.008779	0.002195	2.48	0.045
jP	10	0.058699	0.058699	0.005870	6.64	0.000
*bMax*tP*	24	0.008127	0.008127	0.000339	0.38	0.997
*bMax*jP*	60	0.046617	0.046617	0.000777	0.88	0.720
*tP*jP*	40	0.024818	0.024818	0.000620	0.70	0.911
Error	240	0.212231	0.212231	0.000884		
Total	384	0.365839				

Similar to IGT_{ALL}, we present a full factorial design for the parameter settings of VBIH. The algorithm has three parameters; the maximum block size (*bMax*), the temperature adjustment parameter (*tP*), and the local search jumping probability (*jP*). We consider seven levels for *bMax*, i.e., $bMax \in (2, 3, 4, 5, 6, 7, 8)$; five levels for τP, i.e., $\tau P \in (0.1, 0.2, 0.3, 0.4, 0.5)$; and eleven levels for *jP*, i.e., $jP \in (0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)$, resulting in $7 \times 5 \times 11 = 385$ treatments, as in IGT_{ALL}. The ARPD values of each treatment are determined employing the same method mentioned above, and the ANOVA results are given in Table 3. Before applying ANOVA, three main hypothesis for residuals are tested by making a residual analysis. The results of this analysis show that all three hypothesis could be accepted.

Table 3 indicates that the *jP* and τP are statistically significant as their *p*-values are less than the significance level $\alpha = 0.05$. The parameter *jP* is the most significant factor since it has the greatest *F*-ratio. On the other hand, different levels of *bMax* do not cause statistically significant differences in ARPD, indicating that the VBIH algorithm seems to be robust with regard to *bMax*. There is no statistically significant interaction effect between parameters, therefore the main effects plots of the parameters (Fig. 13) are used to determine the best parameter levels for VBIH.

As also evident from ANOVA, the most significant factor is *jP* for the VBIH algorithm. As shown in Fig. 13, low *jP* values generally result in better ARPD values, which indicates that RSS is more effective than RIS for VBIH. Similar to IGT_{ALL}, a setting of $jP = 0.4$ has better ARPD value than others, therefore this parameter is set as 0.4. The τP parameter has the second largest *F*-ratio and high τP levels result in better ARPD values. Therefore, τP is set as 0.5.

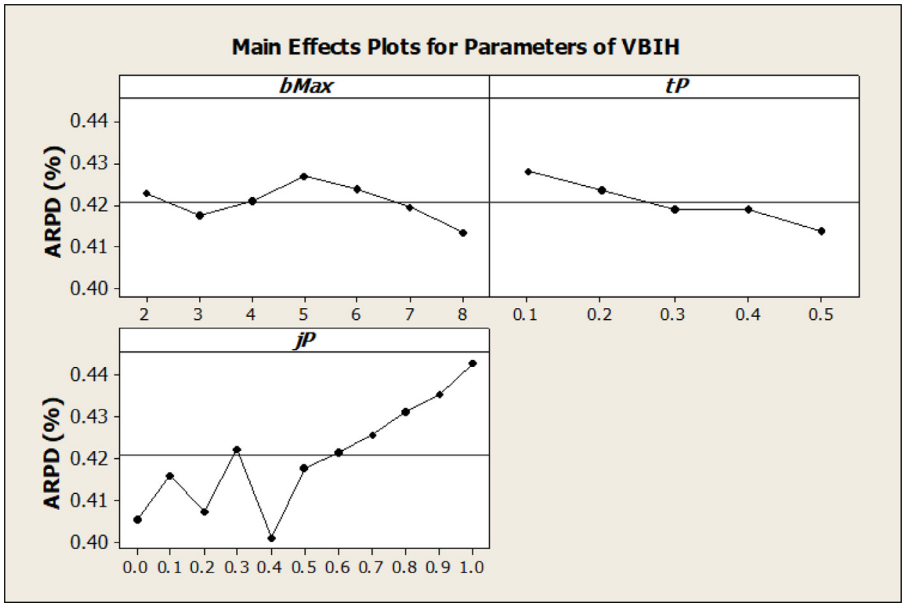

Fig. 13. Main effects plot for parameters of VBIH.

Although there is no statistically significant difference between different levels of *bMax*, $bMax = 8$ provides a slightly better ARPD value than others. Consequently, we set the parameters of VBIH as $bMax = 8$, $\tau P = 0.5$ and $jP = 0.4$.

Finally, based on the aforementioned DOE analyses, the following parameter values are determined for the TFT criterion: $dS = 2$ and $\tau P = 0.5$ in IG_{RS} and IG_{GR}; $dS = 2$, $\tau P = 0.5$ and $jP = 0.4$ in IGT and IGT_{ALL}; $bMax = 8$, $\tau P = 0.5$ and $jP = 0.4$ in VBIH.

6. Computational results

The well-known HFSP benchmark problems provided by Carlier and Neron (2000) are used to evaluate the performance of the proposed IG_{RS}, IG_{GR}, IGT, IGT_{ALL} and VBIH algorithms. In this benchmark set, each problem is labeled with number of jobs, number of stages and the machine layout. The type of the machine layout is denoted by letters *a, b, c*, and *d*, where *c* and *d* layouts are harder to solve:

a: There is single machine in middle stage, and there are 3 machines in other stages.

b: There is single machine in first stage, and there are 3 machines in other stages.

c: There are two machines in middle stage, and 3 machines in other stages.

d: There are 3 machines in all stages.

As an example, a HFSP benchmark with 10 jobs ($j10$), 10 stages ($c10$) and *b* type layout is labeled as $j10c10b3$, where the last digit indicates the problem index. Originally, there are 77 problems in this set: 23 problems with 10 jobs and 5 stages ($j10c5$); 18 problems with 10 jobs and 10 stages ($j10c10$); 24 problems with 15 jobs and 5 stages ($j15c5$); 12 problems with 15 jobs and 10 stages ($j15c10$). As mentioned in Section 2, Liao et al. (2012) proposed additional 10 large instances with 30 jobs and 5 stages ($j30c5e$), where the number of machines in each stage is randomly generated between 3 and 5, and the processing times of jobs are uniform in the range [1,100]. However, these instances are not enough to compare the performances of the metaheuristics on large instances. Therefore, we generate instances with 40, 50 and 60 jobs using the same method (Liao et al., 2012), where there are 10 instances for each size. These large instances are available as supplementary materials of this study.

The performances of IG_{RS}, IG_{GR}, IGT, IGT_{ALL} and VBIH are evaluated by comparisons with time-limited results of the mathematical model. Furthermore, the proposed algorithms are compared with the Discrete Artificial Bee Colony (DABC) algorithm (Pan et al., 2014) and the Improved Discrete Artificial Bee Colony (IDABC) algorithm (Cui and Gu, 2015), which are the two best performing algorithms for the HFSP with makespan criterion. All algorithms including DABC and IDABC are tested on both small and large instances with the total flow time criterion, while, IGT, IGT_{ALL}, VBIH, DABC and IDABC are tested on only large instances with the makespan criterion. The MIP formulation in Section 3 is coded on the IBM ILOG CPLEX Optimization Studio platform, and solved with CPLEX 12.8. The model results are obtained for all instances with a 3-hour time limit on a Core i5, 3.20 GHz, 8 GB RAM computer. All algorithms are coded in C++ programming language on Microsoft Visual Studio 2013, and run on the same configuration. For each algorithm, twenty independent replications are carried out for each instance. For benchmarks in Carlier and Neron (2000), all algorithms are run for $10nm$ milliseconds in each replication, where *n* denotes the number of jobs and *m* represents the number of stages. For large instances with 30, 40, 50 and 60 jobs, all algorithms are run for 50 s in each replication.

The design of experiment explained in Section 5 is also conducted to determine the parameters of IGT_{ALL} and VBIH with the makespan criterion, following the same methodology. Based on this experiment, the following parameter values are set for the C_{max} criterion: $dS = 4$, $\tau P = 0.2$ and $jP = 0.3$ in IGT and IGT_{ALL}; $bMax = 6$, $\tau P = 0.2$ and $jP = 0.3$ in VBIH. The parameters of DABC and IDABC are directly taken from the literature (Pan et al., 2014; Cui and Gu, 2015).

6.1. Comparison of constructive heuristics based on TFT criterion

Before proceeding to the computational results, we first analyze the impact of the proposed GRASP_NEH(x) heuristic on solution quality. As mentioned before, IG_{GR}, IGT, IGT_{ALL} and VBIH employ GRASP_NEH(x) to generate the initial solution, whereas the traditional NEH is used in IG_{RS} for this purpose. We compare the performances of these two constructive heuristics on large instances in terms of TFT. The NEH heuristic is run for a single replication,

Table 4
Comparison of NEH and GRASP_NEH(x) on large instances based on TFT criterion.

Instance	NEH		GRASP_NEH(***x***)				Instance	NEH		GRASP_NEH(***x***)			
	RPD (%)	CPU (ms)	RPD (%)			Avg. CPU (ms)		RPD (%)	CPU (ms)	RPD (%)			Avg. CPU (ms)
			Avg	Min	Max					Avg	Min	Max	
j30c5e1	4.20	0.00	1.87	1.21	2.58	46.10	j50c5e1	7.13	15.00	3.33	2.92	3.73	318.75
j30c5e2	9.31	0.00	2.75	1.93	3.53	46.10	j50c5e2	4.79	0.00	2.71	2.33	2.92	317.95
j30c5e3	9.91	0.00	2.61	1.83	3.11	44.55	j50c5e3	7.86	0.00	2.53	2.14	2.83	312.50
j30c5e4	5.62	0.00	3.13	2.41	3.57	42.15	j50c5e4	7.35	16.00	4.05	3.69	4.59	329.65
j30c5e5	3.90	0.00	2.34	1.79	3.01	45.35	j50c5e5	7.16	15.00	2.87	2.07	3.33	320.30
j30c5e6	5.73	0.00	2.01	1.66	2.55	39.85	j50c5e6	9.12	0.00	3.12	2.76	3.52	335.95
j30c5e7	7.53	0.00	3.07	2.50	3.60	44.50	j50c5e7	8.74	0.00	3.42	3.01	3.86	326.50
j30c5e8	9.98	0.00	3.51	2.70	4.39	42.95	j50c5e8	9.93	0.00	3.81	2.72	4.73	339.10
j30c5e9	8.92	0.00	2.97	2.30	3.50	43.75	j50c5e9	8.08	0.00	2.26	1.75	2.81	365.60
j30c5e10	10.16	0.00	2.59	1.90	3.13	46.10	j50c5e10	8.06	16.00	3.80	3.23	4.53	362.50
j40c5e1	10.56	0.00	2.73	2.20	3.18	143.00	j60c5e1	6.94	0.00	2.83	2.06	3.49	685.90
j40c5e2	11.31	0.00	4.49	3.03	5.33	141.40	j60c5e2	11.45	15.00	2.95	2.64	3.25	701.60
j40c5e3	4.83	15.00	2.08	1.26	2.68	142.20	j60c5e3	7.93	16.00	2.92	2.01	3.73	717.95
j40c5e4	12.34	0.00	4.07	3.48	4.59	141.40	j60c5e4	7.14	16.00	3.22	2.51	3.77	760.15
j40c5e5	13.42	16.00	4.00	3.05	4.83	142.15	j60c5e5	15.92	15.00	3.70	2.86	4.48	702.40
j40c5e6	4.02	0.00	2.31	1.95	2.66	138.25	j60c5e6	16.60	16.00	3.59	3.26	4.02	758.60
j40c5e7	4.90	0.00	2.51	2.13	2.93	140.65	j60c5e7	6.39	16.00	3.01	2.00	3.51	690.60
j40c5e8	2.25	15.00	2.44	1.73	2.88	129.75	j60c5e8	3.60	15.00	2.17	2.00	2.32	691.40
j40c5e9	6.99	0.00	2.86	2.25	3.28	130.45	j60c5e9	3.86	16.00	1.82	1.61	2.01	651.60
j40c5e10	10.59	16.00	3.14	2.51	3.6	132.00	j60c5e10	6.94	15.00	2.19	1.74	2.60	719.50
							Average	**8.04**	**6.60**	**2.94**	**2.33**	**3.47**	**305.78**

while GRASP_NEH(x) is run for 20 independent replications, as it includes a stochastic component.

Table 4 reports the results for each constructive heuristic. In the table, the relative percentage deviation (RPD) between the solution TFT and TFT_{Best} is calculated as follows:

$$RPD = \frac{TFT - TFT_{Best}}{TFT_{Best}} * 100, \quad (13)$$

where TFT_{Best} is the best result obtained by the proposed metaheuristics (reported in Table 7). The average RPD values over 20 replications are reported for GRASP_NEH(x) as well as the maximum and the minimum. Average computation times over 20 replications (Avg. CPU) are also reported for GRASP_NEH(x). As only a single replication is made for NEH, the total CPU time is reported for each instance. Zero values in CPU times indicate negligible solution times.

As shown in Table 4, NEH is very fast with a 6.60 ms average CPU time. However, its average RPD is 8.04%. Even though GRASP_NEH(x) takes 305.78 ms on average, its average RPD is 2.94% from the best-known solutions reported in Table 7. It is clear from the Table 4 that GRASP_NEH(x) substantially outperforms the traditional NEH heuristic by a large margin. In order to justify this judgment, we provide the interval graph of both constructive heuristics in Fig. 14, revealing that GRASP_NEH(x) is significantly better than NEH, as their confidence intervals do not coincide.

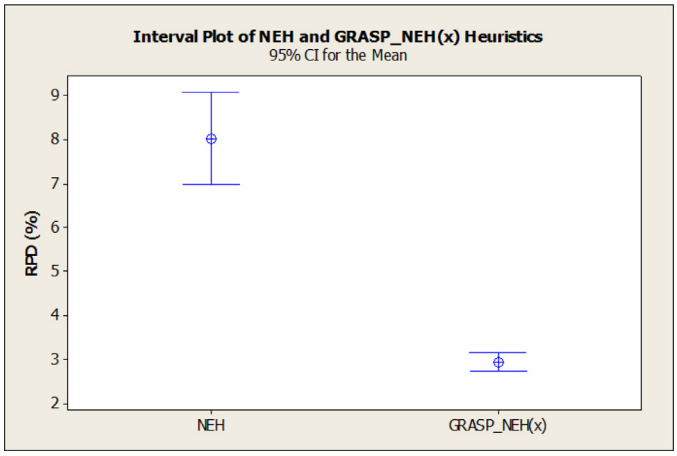

Fig. 14. Interval plot of constructive heuristics.

6.2. Comparison based on TFT criterion

In this subsection, IG_{RS}, IG_{GR}, IGT, IGT_{ALL}, VBIH, DABC, IDABC and time-limited CPLEX are compared for the total flow time (TFT) criterion, using the instances of Carlier and Neron (2000) and the generated large instances. Tables 5–7 report the TFT results for instances with 10 jobs (Carlier and Neron, 2000), instances with 15 jobs (Carlier and Neron, 2000) and large instances, respectively. In the tables, the relative percentage deviation (RPD) between the solution TFT and TFT_{Best} is calculated as in Eq. (13), where TFT_{Best} is the best result among all solution approaches including the CPLEX results. The average RPD values over 20 replications are reported for each algorithm, as well as the maximum and the minimum values. The average computation time (Avg. CPU), in which the algorithm finds the final solutions, is also reported. The time-limited CPLEX bounds are given in all tables. Percent optimality gaps of CPLEX are reported in Tables 5 and 6, while they are not given in Table 7 as the gaps are impracticably high for large instances.

As shown in Table 5, the optimal results are obtained by CPLEX for 26 instances of Carlier and Neron (2000), which are shown in bold. Each algorithm except DABC finds the optimal results for 18 of these 26 instances, while DABC finds the optimal results for 20. Note that the average optimality gap of CPLEX is very low (2%) for instances with 10 jobs. Each algorithm finds TFT_{Best} values for 28 out of 41 instances. The overall performances are very similar for all algorithms, as their average RPDs are same (0.06%) except that it is 0.07% for IG_{RS} and IG_{GR}, and it is 0.08% for DABC. The rest of the algorithms seem to perform slightly better than the DABC due to their smaller average, minimum and maximum RPDs. For the average computation time in which an algorithm finds the final solutions, IG_{RS}, IG_{GR}, IGT, IGT_{ALL} and VBIH are faster than others, as their average CPU times are around 0.01 s. In summary, all proposed algorithms perform very well for small instances due to their fast computational times and very small RPDs.

Table 5
Comparison on instances with 10 jobs (Carlier and Neron, 2000) based on TFT criterion.

Instance	IG_{RS}				IG_{GR}				IGT_{ALL}				IGT				VBIH				DABC				IDABC				CPLEX		
	RPD(%)			Avg. CPU(s)	RPD(%)			Avg. CPU(s)	RPD(%)			Avg. CPU(s)	RPD(%)			Avg. CPU(s)	RPD(%)			Avg. CPU(s)	RPD(%)			Avg. CPU(s)	RPD(%)			Avg. CPU (s)	TFT	Opt. Gap	TFT_{BEST}
	Avg	Min	Max		Avg	Min	Max		Avg	Min	Max		Avg	Min	Max		Avg	Min	Max		Avg	Min	Max		Avg	Min	Max				
j10c5a2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.01	**575**	0.00	575
j10c5a3	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.00	0.00	0.00	0.01	0.13	0.13	0.13	0.00	**749**	0.00	749
j10c5a4	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.00	0.00	0.00	0.02	0.13	0.13	0.13	0.00	**787**	0.00	787
j10c5a5	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.02	0.13	0.13	0.13	0.02	**778**	0.00	778
j10c5a6	0.55	0.55	0.55	0.00	0.55	0.55	0.55	0.00	0.55	0.55	0.55	0.00	0.55	0.55	0.55	0.00	0.55	0.55	0.55	0.00	0.32	0.14	0.55	0.01	0.55	0.55	0.55	0.00	**728**	0.00	728
j10c5b1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	**815**	0.00	815
j10c5b2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	**701**	0.00	701
j10c5b3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	**731**	0.00	731
j10c5b4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	**836**	0.00	836
j10c5b5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	**977**	0.00	977
j10c5b6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	**782**	0.00	782
j10c5c1	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.09	**526**	0.00	526
j10c5c2	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	**567**	0.00	567
j10c5c3	0.19	0.19	0.19	0.00	0.19	0.19	0.19	0.00	0.19	0.19	0.19	0.00	0.19	0.19	0.19	0.00	0.19	0.19	0.19	0.00	0.00	0.00	0.00	0.05	0.19	0.19	0.19	0.00	**539**	0.00	539
j10c5c4	0.19	0.19	0.19	0.01	0.19	0.19	0.19	0.00	0.19	0.19	0.19	0.00	0.19	0.19	0.19	0.00	0.19	0.19	0.19	0.00	0.95	0.95	0.95	0.03	0.19	0.19	0.19	0.00	527	0.11	527
j10c5c5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.34	0.34	0.34	0.04	0.00	0.00	0.00	0.01	**593**	0.00	593
j10c5c6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.01	**530**	0.00	530
j10c5d1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.01	506	0.06	505
j10c5d2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.02	579	0.07	579
j10c5d3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	517	0.10	517
j10c5d4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.17	0.17	0.17	0.03	0.00	0.00	0.00	0.00	573	0.10	572
j10c5d5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	538	0.06	538
j10c5d6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.01	**497**	0.00	497
j10c10a1	0.19	0.19	0.19	0.08	0.19	0.19	0.19	0.07	0.19	0.19	0.19	0.00	0.19	0.19	0.19	0.08	0.19	0.19	0.19	0.01	0.12	0.09	0.65	0.13	0.19	0.19	0.19	0.04	1080	0.001	1080
j10c10a2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	**1193**	0.00	1193
j10c10a3	0.38	0.38	0.38	0.04	0.38	0.38	0.38	0.03	0.38	0.38	0.38	0.03	0.38	0.38	0.38	0.01	0.38	0.38	0.38	0.04	0.47	0.47	0.47	0.03	0.38	0.38	0.38	0.22	**1064**	0.00	1064
j10c10a4	0.27	0.26	0.43	0.00	0.33	0.26	0.43	0.04	0.26	0.26	0.26	0.00	0.26	0.26	0.26	0.04	0.26	0.26	0.26	0.00	0.09	0.09	0.09	0.05	0.26	0.26	0.26	0.01	**1150**	0.00	1150
j10c10a5	0.09	0.09	0.09	0.00	0.09	0.09	0.09	0.00	0.09	0.09	0.09	0.00	0.09	0.09	0.09	0.00	0.09	0.09	0.09	0.00	0.26	0.26	0.26	0.04	0.09	0.09	0.09	0.01	1159	0.001	1159
j10c10a6	0.19	0.19	0.19	0.00	0.19	0.19	0.19	0.01	0.19	0.19	0.19	0.00	0.19	0.19	0.19	0.00	0.19	0.19	0.19	0.00	0.10	0.10	0.10	0.05	0.19	0.19	0.19	0.00	**1037**	0.00	1037
j10c10b1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	**1210**	0.00	1210
j10c10b2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	**1216**	0.00	1216
j10c10b3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	**1272**	0.00	1272
j10c10b4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	1238	0.001	1238
j10c10b5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	**1280**	0.00	1280
j10c10b6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	**1250**	0.00	1250
j10c10c1	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.20	0.20	0.20	0.03	0.00	0.00	0.00	0.01	984	0.02	984
j10c10c2	0.10	0.10	0.10	0.00	0.10	0.10	0.10	0.00	0.10	0.10	0.10	0.00	0.10	0.10	0.10	0.00	0.10	0.10	0.10	0.00	0.00	0.00	0.00	0.07	0.10	0.10	0.10	0.02	1024	0.001	1024
j10c10c3	0.07	0.00	0.10	0.18	0.07	0.00	0.10	0.11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.02	0.03	0.00	0.10	0.34	0.00	0.00	0.00	0.11	1004	0.04	1004
j10c10c4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.01	1051	0.06	1049
j10c10c5	0.09	0.09	0.09	0.01	0.09	0.09	0.09	0.01	0.09	0.09	0.09	0.00	0.09	0.09	0.09	0.00	0.09	0.09	0.09	0.00	0.09	0.09	0.09	0.12	0.09	0.09	0.09	0.01	1061	0.01	1061
j10c10c6	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.17	0.11	0.21	0.06	0.00	0.00	0.00	0.04	954	0.09	947
Average	**0.07**	**0.06**	**0.07**	**0.01**	**0.07**	**0.06**	**0.07**	**0.01**	**0.06**	**0.06**	**0.06**	**0.001**	**0.06**	**0.06**	**0.06**	**0.005**	**0.06**	**0.06**	**0.06**	**0.003**	**0.08**	**0.08**	**0.11**	**0.04**	**0.06**	**0.06**	**0.06**	**0.02**		**0.02**	

Table 6
Comparison on instances with 15 jobs (Carlier and Neron, 2000) based on TFT criterion.

Instance	IG_{RS}				IG_{GR}				IGT_{ALL}				IGT				VBIH				DABC				IDABC				CPLEX		TFT_{BEST}
	RPD(%)			Avg. CPU	RPD(%)			Avg. CPU	RPD(%)			Avg. CPU	RPD(%)			Avg. CPU	RPD(%)			Avg. CPU	RPD(%)			Avg. CPU	RPD(%)			Avg. CPU	TFT	Opt. Gap	
	Avg	Min	Max	(s)	Avg	Min	Max	(s)	Avg	Min	Max	(s)	Avg	Min	Max	(s)	Avg	Min	Max	(s)	Avg	Min	Max	(s)	Avg	Min	Max	(s)			
j15c5a1	0.07	0.07	0.07	0.00	0.07	0.07	0.07	0.00	0.07	0.07	0.07	0.00	0.07	0.07	0.07	0.00	0.07	0.07	0.07	0.00	0.00	0.00	0.00	0.09	0.07	0.07	0.07	0.02	1505	0.25	1505
j15c5a2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.02	1358	0.24	1358
j15c5a3	0.10	0.10	0.10	0.02	0.10	0.10	0.10	0.03	0.10	0.10	0.10	0.02	0.10	0.10	0.10	0.02	0.10	0.10	0.10	0.02	0.01	0.00	0.10	0.08	0.10	0.10	0.10	0.12	1050	0.18	1050
j15c5a4	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.02	1274	0.19	1274
j15c5a5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.02	1382	0.22	1382
j15c5a6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.02	1583	0.28	1583
j15c5b1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	1473	0.25	1473
j15c5b2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	1310	0.29	1310
j15c5b3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	1368	0.13	1368
j15c5b4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	1259	0.19	1257
j15c5b5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	1432	0.27	1432
j15c5b6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.01	1580	0.17	1580
j15c5c1	0.03	0.00	0.22	0.11	0.01	0.00	0.11	0.10	0.00	0.00	0.00	0.03	0.01	0.00	0.11	0.03	0.00	0.00	0.00	0.03	0.04	0.00	0.33	0.22	0.01	0.00	0.11	0.18	905	0.26	900
j15c5c2	0.18	0.00	0.51	0.10	0.20	0.00	0.51	0.16	0.01	0.00	0.10	0.18	0.11	0.00	0.41	0.10	0.01	0.00	0.10	0.12	0.08	0.00	0.31	0.15	0.07	0.00	0.41	0.20	996	0.26	977
j15c5c3	0.09	0.00	0.10	0.13	0.06	0.00	0.10	0.10	0.00	0.00	0.00	0.19	0.03	0.00	0.10	0.18	0.01	0.00	0.10	0.17	0.08	0.00	0.10	0.16	0.06	0.00	0.10	0.17	975	0.30	954
j15c5c4	0.01	0.00	0.11	0.09	0.01	0.00	0.11	0.05	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.11	0.13	0.00	0.00	0.00	0.08	920	0.26	912
j15c5c5	0.00	0.00	0.00	0.14	0.01	0.00	0.12	0.14	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.08	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.18	0.00	0.00	0.00	0.11	854	0.26	831
j15c5c6	0.00	0.00	0.00	0.16	0.17	0.00	0.89	0.13	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.03	0.11	0.00	0.40	0.36	0.00	0.00	0.00	0.25	1033	0.25	1012
j15c5d1	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.02	1295	0.20	1295
j15c5d2	0.00	0.00	0.00	0.08	0.00	0.00	0.00	0.08	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.05	0.01	0.00	0.10	0.26	0.01	0.00	0.10	0.16	983	0.28	958
j15c5d3	0.02	0.00	0.11	0.22	0.01	0.00	0.11	0.25	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.03	0.04	0.00	0.11	0.28	0.01	0.00	0.11	0.16	953	0.26	931
j15c5d4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.00	0.00	0.00	0.02	910	0.23	905
j15c5d5	0.00	0.00	0.00	0.11	0.01	0.00	0.11	0.09	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.26	0.00	0.00	0.00	0.09	906	0.24	888
j15c5d6	0.07	0.00	0.22	0.27	0.08	0.00	0.22	0.24	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.32	0.01	0.00	0.11	0.18	0.04	0.00	0.22	0.28	0.08	0.00	0.22	0.23	932	0.28	903
j15c10a1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.00	0.00	0.00	0.47	0.00	0.00	0.00	0.19	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	2422	0.22	2422
j15c10a2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.07	0.00	0.00	0.00	0.02	2129	0.16	2128
j15c10a3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	2156	0.13	2156
j15c10a4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	2250	0.21	2250
j15c10a5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.03	1901	0.12	1901
j15c10a6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00	2113	0.17	2113
j15c10b1	0.00	0.00	0.00	0.07	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.23	0.00	0.00	0.00	0.11	2053	0.13	2052
j15c10b2	0.14	0.11	0.76	0.23	0.11	0.11	0.11	0.08	0.11	0.11	0.11	0.06	0.11	0.11	0.11	0.03	0.11	0.11	0.11	0.03	0.11	0.11	0.11	0.44	0.11	0.11	0.11	0.15	1839	0.13	1839
j15c10b3	0.25	0.05	0.84	0.30	0.20	0.05	0.84	0.32	0.05	0.05	0.09	0.54	0.07	0.05	0.19	0.61	0.07	0.05	0.28	0.60	0.11	0.00	0.52	0.78	0.17	0.05	0.52	0.77	2131	0.16	2131
j15c10b4	0.34	0.29	0.39	0.56	0.33	0.29	0.58	0.70	0.30	0.29	0.39	0.56	0.32	0.29	0.39	0.27	0.31	0.29	0.39	0.36	0.36	0.24	0.53	0.50	0.32	0.29	0.39	0.61	2068	0.16	2068
j15c10b5	0.11	0.00	0.21	0.14	0.10	0.00	0.21	0.22	0.08	0.00	0.11	0.20	0.08	0.00	0.11	0.13	0.03	0.00	0.11	0.33	0.04	0.00	0.05	0.38	0.05	0.00	0.11	0.39	1864	0.12	1864
j15c10b6	0.05	0.05	0.05	0.11	0.05	0.05	0.05	0.13	0.05	0.05	0.05	0.04	0.05	0.05	0.05	0.04	0.05	0.05	0.05	0.05	0.02	0.00	0.10	0.38	0.05	0.05	0.05	0.18	1999	0.11	1999
Average	**0.04**	**0.02**	**0.10**	**0.08**	**0.04**	**0.02**	**0.12**	**0.08**	**0.02**	**0.02**	**0.03**	**0.06**	**0.03**	**0.02**	**0.05**	**0.07**	**0.02**	**0.02**	**0.04**	**0.06**	**0.03**	**0.01**	**0.09**	**0.17**	**0.03**	**0.02**	**0.07**	**0.12**		**0.21**	

Table 7
Comparison on large instances based on TFT criterion.

Instance	IG_{RS}				IG_{GR}				IGT_{ALL}				IGT				VBIH				DABC				IDABC				CPLEX		TFT_{BEST}
	RPD(%)			Avg. CPU (s)	RPD(%)			Avg. CPU (s)	RPD(%)			Avg. CPU (s)	RPD(%)			Avg. CPU (s)	RPD(%)			Avg. CPU (s)	RPD(%)			Avg. CPU (s)	RPD(%)			Avg. CPU (s)	TFT	RPD(%)	
	Avg	Min	Max		Avg	Min	Max		Avg	Min	Max		Avg	Min	Max		Avg	Min	Max		Avg	Min	Max		Avg	Min	Max				
j30c5e1	0.11	0.00	0.28	12.30	0.17	0.05	0.36	16.43	0.01	0.00	0.05	16.10	0.04	0.00	0.25	14.67	0.01	0.00	0.08	14.03	0.15	0.00	0.35	17.00	0.11	0.02	0.24	19.78	10,445	7.25	9739
j30c5e2	0.48	0.21	0.97	11.08	0.45	0.17	1.04	17.24	0.21	0.10	0.43	19.79	0.28	0.11	0.55	17.12	0.22	0.00	0.49	24.74	0.20	0.00	0.51	27.70	0.37	0.17	0.52	26.05	12,027	4.91	11,464
j30c5e3	0.41	0.02	0.93	21.23	0.38	0.12	0.72	20.41	0.16	0.07	0.30	18.81	0.22	0.03	0.53	15.32	0.12	0.00	0.34	18.62	0.22	0.05	0.48	31.17	0.21	0.02	0.48	26.77	12,998	6.48	12,207
j30c5e4	0.61	0.04	1.39	13.62	0.44	0.11	0.92	19.40	0.13	0.00	0.70	17.95	0.20	0.00	0.63	17.57	0.10	0.00	0.58	14.60	0.31	0.05	0.75	28.12	0.33	0.00	0.71	24.51	11,633	10.09	10,567
j30c5e5	0.32	0.15	0.54	16.61	0.29	0.08	0.68	17.28	0.12	0.05	0.27	18.32	0.12	0.00	0.25	13.69	0.09	0.00	0.22	20.77	0.15	0.02	0.32	25.89	0.18	0.05	0.37	32.09	12,602	4.80	12,025
j30c5e6	0.08	0.00	0.23	17.07	0.11	0.00	0.29	14.01	0.02	0.00	0.13	13.27	0.07	0.00	0.20	11.28	0.01	0.00	0.13	15.32	0.07	0.00	0.20	20.17	0.07	0.00	0.16	24.75	12,218	4.27	11,718
j30c5e7	0.76	0.17	1.30	16.93	0.66	0.00	1.04	13.98	0.23	0.00	0.51	25.26	0.31	0.00	0.71	12.29	0.23	0.00	0.51	24.32	0.42	0.00	0.85	29.04	0.35	0.09	0.60	30.18	12,189	6.25	11,472
j30c5e8	0.77	0.00	1.22	16.78	0.71	0.30	1.42	19.10	0.37	0.00	0.62	23.30	0.38	0.00	0.66	22.83	0.41	0.12	0.78	16.28	0.46	0.00	0.83	25.22	0.62	0.22	1.06	30.66	12,828	6.34	12,063
j30c5e9	0.63	0.31	1.15	20.51	0.49	0.30	0.83	15.90	0.25	0.00	0.50	18.07	0.36	0.04	0.70	13.10	0.23	0.00	0.50	18.22	0.29	0.00	0.47	30.31	0.41	0.21	0.65	21.66	12,706	4.92	12,110
j30c5e10	0.17	0.00	0.43	24.84	0.14	0.00	0.36	21.61	0.05	0.00	0.13	21.40	0.05	0.00	0.22	19.09	0.03	0.00	0.15	24.96	0.31	0.11	0.78	16.93	0.16	0.00	0.32	27.24	12,318	4.89	11,744
j40c5e1	1.03	0.35	1.97	28.43	0.57	0.18	0.98	23.70	0.18	0.00	0.32	26.47	0.28	0.02	0.58	26.23	0.22	0.01	0.61	30.15	0.36	0.16	0.84	31.05	0.54	0.24	0.86	36.41	17,934	8.70	16,499
j40c5e2	0.95	0.55	1.90	30.20	0.96	0.47	1.42	35.76	0.48	0.00	0.86	31.15	0.57	0.26	1.09	26.81	0.55	0.21	1.00	28.68	0.49	0.28	0.71	34.12	0.84	0.44	1.10	35.64	19,664	13.16	17,377
j40c5e3	0.04	0.00	0.14	29.75	0.05	0.00	0.47	27.82	0.00	0.00	0.03	16.63	0.02	0.00	0.06	20.21	0.01	0.00	0.04	14.38	0.07	0.00	0.24	22.18	0.02	0.00	0.06	21.73	19,360	4.20	18,580
j40c5e4	1.45	0.85	2.06	26.01	1.03	0.34	1.57	19.27	0.61	0.28	0.99	29.71	0.68	0.00	1.18	28.33	0.59	0.25	1.30	30.89	0.65	0.33	1.07	31.95	0.79	0.39	1.14	39.72	17,989	8.42	16,592
j40c5e5	0.89	0.31	1.39	31.80	0.76	0.25	1.55	31.21	0.46	0.06	0.79	29.71	0.53	0.00	1.00	28.55	0.47	0.18	0.93	29.87	0.44	0.22	0.69	35.14	0.72	0.38	0.99	38.06	19,036	11.48	17,076
j40c5e6	0.35	0.18	0.56	25.68	0.24	0.00	0.75	24.87	0.14	0.00	0.46	26.83	0.14	0.01	0.35	23.09	0.13	0.00	0.31	25.84	0.40	0.21	1.05	31.68	0.24	0.10	0.40	36.16	21,037	7.98	19,483
j40c5e7	0.56	0.18	1.17	27.34	0.53	0.19	1.01	29.48	0.24	0.08	0.64	28.42	0.30	0.00	0.58	28.83	0.25	0.04	0.43	29.65	0.32	0.07	0.63	32.76	0.32	0.10	0.57	38.04	18,984	5.22	18,043
j40c5e8	0.26	0.12	0.51	28.52	0.28	0.06	0.49	23.85	0.16	0.08	0.30	28.26	0.19	0.09	0.39	30.45	0.14	0.00	0.23	22.50	0.26	0.12	0.47	37.00	0.20	0.13	0.34	35.24	20,999	6.40	19,736
j40c5e9	0.37	0.19	0.60	27.73	0.39	0.21	0.59	28.45	0.21	0.00	0.37	22.32	0.26	0.15	0.45	32.56	0.20	0.06	0.31	29.76	0.34	0.17	0.46	33.54	0.31	0.16	0.45	35.55	20,464	4.72	19,541
j40c5e10	0.81	0.24	1.44	31.87	0.66	0.16	1.25	31.04	0.37	0.00	0.65	31.45	0.40	0.08	0.95	23.51	0.42	0.11	0.76	24.77	0.56	0.10	1.08	37.33	0.74	0.37	1.11	35.16	19,930	7.63	18,518
j50c5e1	0.98	0.46	1.65	30.19	0.98	0.57	1.28	27.97	0.50	0.17	0.88	38.21	0.61	0.13	0.99	32.89	0.49	0.00	1.03	37.21	0.89	0.53	1.15	35.28	0.74	0.31	0.96	39.92	31,390	13.99	27,538
j50c5e2	0.60	0.30	0.81	26.39	0.53	0.10	0.88	34.92	0.21	0.04	0.41	30.29	0.33	0.05	0.60	25.06	0.22	0.00	0.37	29.62	0.35	0.18	0.58	34.87	0.43	0.27	0.62	39.39	29,247	6.46	27,472
j50c5e3	0.92	0.49	1.42	31.18	0.63	0.44	1.10	32.91	0.28	0.00	0.53	38.78	0.32	0.18	0.43	34.53	0.24	0.09	0.39	36.59	0.57	0.35	0.79	29.72	0.48	0.09	0.65	43.90	31,720	9.33	29,013
j50c5e4	1.10	0.70	1.65	24.79	0.86	0.33	1.30	26.16	0.47	0.00	1.07	35.71	0.64	0.15	1.14	33.17	0.45	0.02	0.80	30.12	0.68	0.40	1.08	32.21	0.78	0.31	1.15	40.31	25,697	9.38	23,494
j50c5e5	0.74	0.46	1.08	27.31	0.68	0.21	1.33	31.90	0.47	0.00	0.96	40.07	0.42	0.11	0.81	31.07	0.42	0.16	0.70	32.05	0.92	0.15	1.87	35.74	0.74	0.45	1.22	42.62	31,076	10.89	28,024
j50c5e6	0.78	0.19	1.46	34.77	0.59	0.22	1.20	34.47	0.24	0.04	0.74	32.09	0.27	0.03	0.56	32.63	0.17	0.00	0.39	28.09	0.55	0.21	1.02	33.19	0.65	0.34	1.12	41.45	27,248	12.09	24,309
j50c5e7	0.86	0.41	1.42	33.31	0.47	0.11	1.10	30.12	0.33	0.11	0.65	38.81	0.31	0.00	0.62	29.76	0.28	0.08	0.47	35.27	0.58	0.24	0.87	31.54	0.69	0.30	1.14	41.82	27,929	9.84	25,427
j50c5e8	1.03	0.34	1.81	33.76	0.69	0.36	1.23	33.97	0.38	0.00	0.70	33.00	0.48	0.12	0.82	34.70	0.45	0.11	0.96	31.99	0.61	0.23	0.93	31.78	1.01	0.47	1.45	42.69	26,096	11.89	23,322
j50c5e9	0.95	0.53	1.43	33.06	0.67	0.28	1.52	31.07	0.28	0.00	0.54	34.46	0.33	0.00	0.94	26.67	0.35	0.12	0.57	30.76	0.61	0.34	1.05	38.30	0.70	0.32	1.06	41.56	22,259	10.09	20,218
j50c5e10	1.11	0.46	1.54	37.30	1.01	0.53	1.78	35.61	0.55	0.24	0.76	38.61	0.58	0.11	1.22	35.01	0.58	0.00	1.03	35.14	1.03	0.44	1.65	29.90	1.04	0.67	1.43	46.60	28,340	11.29	25,465
j60c5e1	1.02	0.65	1.73	36.63	0.82	0.42	1.19	30.52	0.33	0.07	0.54	37.80	0.35	0.00	0.74	34.66	0.37	0.03	0.68	31.76	0.71	0.34	1.80	35.74	0.87	0.46	1.29	45.51	38,697	12.44	34,417
j60c5e2	1.09	0.69	1.89	35.28	0.67	0.13	1.05	38.55	0.31	0.06	0.61	38.43	0.37	0.19	0.56	37.58	0.38	0.00	0.76	38.40	0.78	0.47	1.19	34.78	0.76	0.28	1.17	48.23	36,911	13.41	32,546
j60c5e3	0.80	0.47	1.16	37.68	0.54	0.17	0.86	28.75	0.31	0.05	0.61	38.77	0.34	0.08	0.74	32.90	0.21	0.00	0.56	40.12	0.52	0.23	1.22	36.70	0.69	0.41	0.93	43.28	38,101	10.15	34,590
j60c5e4	1.66	1.12	2.28	39.54	1.00	0.51	1.69	30.89	0.54	0.21	1.02	35.50	0.53	0.00	0.93	31.70	0.53	0.10	1.00	39.45	1.03	0.44	1.80	31.83	1.38	0.94	2.19	46.59	32,875	15.31	28,510
j60c5e5	1.14	0.57	1.74	39.79	0.94	0.30	1.32	38.27	0.56	0.00	1.04	36.36	0.66	0.17	1.19	38.41	0.50	0.17	0.96	39.52	0.87	0.41	1.64	37.69	1.10	0.56	1.50	45.68	38,597	14.47	33,719
j60c5e6	1.33	0.42	2.27	36.50	0.86	0.40	1.36	32.64	0.63	0.28	1.03	38.12	0.71	0.00	1.43	35.24	0.57	0.10	1.07	36.99	0.90	0.41	1.73	38.93	1.36	0.71	2.22	47.91	35,095	15.71	30,331
j60c5e7	0.64	0.37	1.00	36.45	0.52	0.23	0.88	38.41	0.26	0.09	0.38	35.01	0.28	0.06	0.58	36.62	0.22	0.00	0.35	39.12	0.66	0.23	1.13	37.40	0.39	0.29	0.60	48.21	43,981	14.37	38,454
j60c5e8	0.61	0.34	1.11	39.51	0.44	0.27	0.87	33.78	0.21	0.00	0.45	41.19	0.26	0.08	0.44	37.39	0.22	0.06	0.41	39.49	0.47	0.28	0.72	32.04	0.49	0.35	0.71	43.75	36,342	9.01	33,337
j60c5e9	0.40	0.19	0.70	40.28	0.38	0.18	0.61	36.99	0.11	0.02	0.19	36.82	0.13	0.02	0.24	36.32	0.11	0.00	0.27	33.06	0.35	0.14	0.56	32.70	0.25	0.00	0.35	44.68	41,321	12.74	36,650
j60c5e10	1.27	0.83	2.37	35.57	0.77	0.40	1.14	30.71	0.29	0.00	0.68	40.47	0.44	0.06	0.95	33.77	0.35	0.04	0.70	37.64	0.70	0.14	1.16	38.01	0.91	0.68	1.13	43.70	37,208	15.00	32,355
Average	**0.75**	**0.35**	**1.27**	**28.7**	**0.58**	**0.23**	**1.04**	**27.74**	**0.29**	**0.05**	**0.57**	**30.04**	**0.34**	**0.06**	**0.68**	**27.39**	**0.29**	**0.05**	**0.58**	**29.02**	**0.51**	**0.20**	**0.92**	**31.67**	**0.57**	**0.28**	**0.88**	**37.33**		**9.40**	

For instances with 15 jobs, the optimal results cannot be obtained by CPLEX under the 3-hour time limit, and the average optimality gap is 21%. As shown in Table 6, DABC finds TFT_{Best} values for 34 out of 36 instances, while the rest of the algorithms find 30, and CPLEX finds only 22. Similar to instances with 10 jobs, the overall performances are quite similar, as algorithms' average RPDs are between 0.02% and 0.04%. However, IGT_{ALL} and VBIH seem to perform slightly better than the rest. IG_{RS} and IG_{GR} are the worst performing ones as they have the largest average and maximum RPDs. For the average CPU times, even though all algorithms are fast, it is clear that IG_{RS}, IG_{GR}, IGT, IGT_{ALL} and VBIH find the final solutions faster than others. It can be stated that IGT_{ALL} and VBIH are the best performing for these instances due to their faster computational times and smaller RPDs.

For large instances, the optimal results cannot be obtained within the time limit, as expected. As optimality gaps are very high (between 31% and 65%) for these instances, they are not reported in Table 7 and only the RPD values from the best-known solutions are reported. CPLEX cannot find any of the TFT_{Best} values for these 40 large instances, and its average RPD is 9.40%. All heuristic algorithms clearly outperform the time-limited CPLEX.

When the heuristics are compared with each other in terms of the number of TFT_{Best} values obtained, IG_{RS} finds 5 TFT_{Best} values, whereas IG_{GR} finds 5, IGT_{ALL} finds 20, IGT finds 16, VBIH finds 20, DABC finds 7, and IDABC obtains 5. As shown in Table 7, VBIH and IGT_{ALL} are the best performers with 0.29% average RPD, smaller than those generated by IGT (0.34%), DABC (0.51%), IDABC (0.57%), IG_{GR} (0.58%) and IG_{RS} (0.75%). Note that IGT also has small RPDs, while DABC, IDABC IG_{GR} and IG_{RS} have much larger RPDs. Especially, IG_{RS} is the worst performing as it has the largest RPD values. In terms of average CPU times, the performances are similar for IG_{RS}, IG_{GR}, IGT, IGT_{ALL} and VBIH, as they find the final solutions between 27 and 30 s. DABC and IDABC find the final solutions slower than the rest of the heuristics. In summary, it can be said that the VBIH and IGT_{ALL} are the best performing among all due to their higher number of TFT_{Best} values found and small RPDs. However, it is worth to mention that IGT also has a very good performance in terms of TFT_{Best} and RPD. DABC, IDABC, IG_{GR} and IG_{RS} perform much worse than the rest in terms of the same criteria.

In order to test performance of the algorithms on large instances statistically, we carried out a series of Wilcoxon signed-rank tests at the significance level $\alpha = 0.05$, based on the results in Table 7. Wilcoxon signed-rank test is a non-parametric test to compare two related samples, which is based on differences between paired observations. We use this test to determine whether there is a statistically significant difference in average RPDs between the two algorithms. As we have 40 large instances, the sample size is 40 for each pairwise comparison of algorithms. Let m_D denotes the median of the difference between the average RPDs generated by two different algorithms, the null hypothesis is defined by H_0: $m_D = 0$ saying that there is no difference between the average RPDs of two algorithms compared. On the other hand, the alternative hypothesis is defined by H_1: $m_D \neq 0$ saying that there is a difference between the average RPDs generated by two algorithms compared.

The Wilcoxon signed-rank test results for the pairwise comparisons of algorithms are presented in Table 8. As shown in the table, all the pairwise differences are statistically significant at the $\alpha = 0.05$ level, except IG_{GR} vs IDABC and IGT_{ALL} vs VBIH pairs. For these pairs, the differences between the algorithms are not meaningful at the $\alpha = 0.05$ level, implying that algorithms within these pairs have similar performances.

We also provide the interval plot of all algorithms in Fig. 15. As shown in Fig. 15, IGT, IGT_{ALL} and VBIH are statistically equivalent as their confidence intervals coincide. But, they are all significantly and statistically better than DABC, IDABC, IG_{GR} and IG_{RS}. As shown in Fig. 15, DABC, IDABC and IG_{GR} are statistically equivalent as their confidence intervals overlap. Note that, DABC is also significantly better than IG_{RS} statistically.

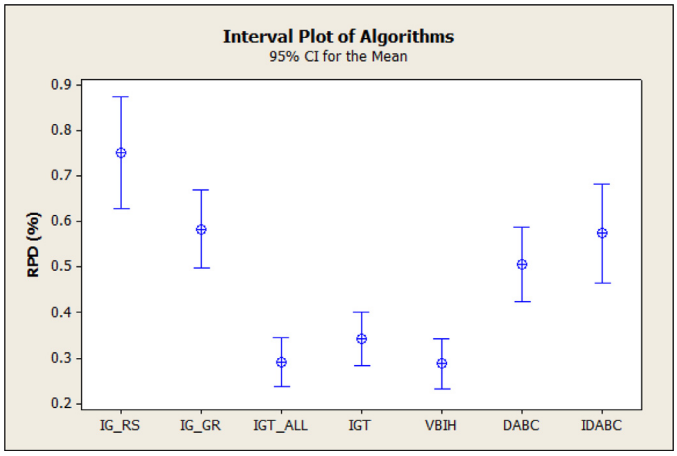

Fig. 15. Interval plot of algorithms based on TFT criterion.

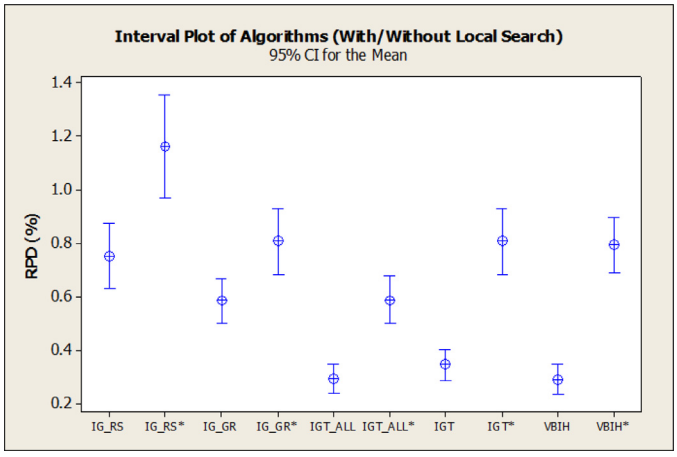

Fig. 16. Interval plot of algorithms (with/without local search).

As commented above, IGT, IGT_{ALL} and VBIH perform significantly better than others. One of the reasons of their performance is that all of these algorithms employ GRASP_NEH(x) as the constructive heuristic instead of the standard NEH procedure. As mentioned in Section 6.1, GRASP_NEH(x) significantly outperforms NEH in terms of solution quality. This conclusion can also be verified by comparing the performances of IG_{GR} and IG_{RS}. Note that, the only difference between IG_{GR} and IG_{RS} is that IG_{GR} uses GRASP_NEH(x) as the constructive heuristic instead of NEH. As IG_{GR} performs better than the traditional IG_{RS}, it is evident that using GRASP_NEH(x) to generate initial solutions improves the performance of the algorithm.

Another common feature of IGT, IGT_{ALL} and VBIH is that all employ RIS and RSS after the construction stage. These local searches might also have an effect on performance. In order to test this effect, we run the algorithms without RIS/RSS and compare the performances. We also make the same comparison for IG_{RS}, which employs first-improvement insertion local search after construction. We provide the interval plots for the two versions (with and without local search) of the algorithms (IG_{RS}, IG_{GR}, IGT_{ALL}, IGT and VBIH) in Fig. 16. The algorithms without the local searches are marked with asterisks in the figure. It seems that applying local search after construction significantly improves algorithm performance statistically.

Table 8
Wilcoxon signed-rank test results for large instances based on TFT criterion.

Pairs of algorithms	p-value	Significant? ($p \leq 0.05$)	Decision	Pairs of algorithms	p-value	Significant? ($p \leq 0.05$)	Decision
IG_{RS} vs IG_{GR}	0.000	Yes	Reject H_0	IGT_{ALL} vs IGT	0.000	Yes	Reject H_0
IG_{RS} vs IGT_{ALL}	0.000	Yes	Reject H_0	IGT_{ALL} vs VBIH	0.559	No	Fail to reject H_0
IG_{RS} vs IGT	0.000	Yes	Reject H_0	IGT_{ALL} vs DABC	0.000	Yes	Reject H_0
IG_{RS} vs VBIH	0.000	Yes	Reject H_0	IGT_{ALL} vs IDABC	0.000	Yes	Reject H_0
IG_{RS} vs DABC	0.000	Yes	Reject H_0	IGT vs VBIH	0.000	Yes	Reject H_0
IG_{RS} vs IDABC	0.000	Yes	Reject H_0	IGT vs DABC	0.000	Yes	Reject H_0
IG_{GR} vs IGT_{ALL}	0.000	Yes	Reject H_0	IGT vs IDABC	0.000	Yes	Reject H_0
IG_{GR} vs IGT	0.000	Yes	Reject H_0	VBIH vs DABC	0.000	Yes	Reject H_0
IG_{GR} vs VBIH	0.000	Yes	Reject H_0	VBIH vs IDABC	0.000	Yes	Reject H_0
IG_{GR} vs DABC	0.003	Yes	Reject H_0	DABC vs IDABC	0.021	Yes	Reject H_0
IG_{GR} vs IDABC	0.285	No	Fail to reject H_0				

Table 9
Comparison on instances with 30 jobs (Liao et al., 2012) based on C_{max} criterion.

Instance	IGT_{ALL}				IGT				VBIH				C_{MAX} (Cui and Gu, 2015)	BC_{MAX}
	RPD(%)			Avg. CPU	RPD(%)			Avg. CPU	RPD(%)			Avg. CPU		
	Avg	Min	Max	(s)	Avg	Min	Max	(s)	Avg	Min	Max	(s)		
j30c5e1	0.57	0.22	0.87	17.84	0.26	0.00	0.43	14.94	0.29	0.22	0.43	21.22	463	**462**
j30c5e2	0.00	0.00	0.00	0.43	0.00	0.00	0.00	0.21	0.00	0.00	0.00	0.40	616	616
j30c5e3	0.50	0.00	0.84	19.80	0.25	0.00	0.51	22.69	0.41	0.17	0.84	23.41	593	593
j30c5e4	0.67	0.36	1.07	23.82	0.36	0.00	0.53	19.40	0.44	0.36	0.53	19.11	565	**563**
j30c5e5	0.36	0.00	0.50	16.14	0.27	0.00	0.50	22.60	0.35	0.00	0.50	16.02	600	600
j30c5e6	0.17	0.00	0.33	14.80	0.11	0.00	0.17	12.45	0.17	0.00	0.50	22.86	601	**600**
j30c5e7	0.00	0.00	0.00	5.77	0.00	0.00	0.00	1.58	0.00	0.00	0.00	2.47	626	626
j30c5e8	0.00	0.00	0.00	19.71	0.00	0.00	0.00	7.07	0.00	0.00	0.00	6.55	674	674
j30c5e9	0.31	0.00	0.47	19.41	0.08	0.00	0.16	15.90	0.05	0.00	0.16	25.27	642	642
j30c5e10	0.90	0.17	1.22	27.20	0.49	0.00	0.87	23.92	0.64	0.17	1.05	21.46	573	573
Average	**0.35**	**0.07**	**0.53**	**16.49**	**0.18**	**0.00**	**0.32**	**14.08**	**0.23**	**0.09**	**0.40**	**15.88**		

The effectiveness of RIS and RSS against first-improvement insertion local search can also be verified by comparing the performances of IG_{GR} and IGT. Note that, the only difference between IG_{GR} and IGT is that IGT employs RIS and RSS as the local search instead of first-improvement insertion local search. As IGT clearly performs better than the IG_{GR}, it can be said that using RIS/RSS as local search significantly improves the performance of the algorithm.

When we focus on these three best performing heuristics, IGT_{ALL} and VBIH perform slightly better than IGT. Note that IGT_{ALL} is IGT plus an extra local search to partial solutions after destruction. The additional local search clearly improves solution quality. The same idea is also employed in VBIH, as it applies local search to the partial solutions after removing the block. Consequently, it is evident that applying a local search to partial solutions before construction substantially improves the performance of the algorithms.

As a result, three key components of IGT_{ALL} and VBIH cause these algorithms to perform better than others: (i) using GRASP_NEH(x) as the constructive heuristic, (ii) applying local search to partial solutions after destruction, and (iii) employing RIS and RSS local search with a probability after construction.

6.3. Comparison based on C_{max} criterion

In this subsection, we test the performances of IGT, IGT_{ALL}, VBIH, DABC and IDABC algorithms on large instances for the makespan (C_{max}) criterion. Our aim is to provide the best-known solutions for C_{max} for the new benchmark suite. Tables 9 and 10 report the results for instances with 30 jobs (Liao et al., 2012) and other large instances, respectively. The relative percentage deviation (RPD) between the solution C_{max} and BC_{max} is calculated as follows:

$$RPD = \frac{C_{max} - BC_{max}}{BC_{max}} * 100 \quad (14)$$

where BC_{max} is the objective value of the best solution. The average RPD values over 20 replications are reported for each algorithm, as well as the maximum and the minimum values. The average computation time (Avg. CPU), in which the algorithm finds the final solutions, is also reported for each instance. Time-limited CPLEX results are given in Table 10, as well as their RPDs.

As the current best-known C_{max} values are reported in literature for instances with 30 jobs (Cui and Gu, 2015), these results are included in Table 9. Note that, IDABC was run for twenty replications for each instance with a time limit of 200 s in Cui and Gu (2015). When IGT, IGT_{ALL} and VBIH are compared with each other in terms of the number of best solutions found, IGT_{ALL} finds 7, IGT finds 10 and VBIH finds 6 of the BC_{max} values out of 10 instances. Note that IGT finds all BC_{max} values for the 10 instances. In terms of average RPDs, IGT (0.18%) and VBIH (0.23%) perform better than the IGT_{ALL} (0.35%). In overall average, IGT is the best performer with 0.00% minimum RPD, which is smaller than IGT_{ALL} (0.07%) and VBIH (0.09%) minimums. In terms of average CPU times, the performances are similar for all algorithms as they obtain final solutions between 14 and 16.5 s. However, IGT seems to be slightly faster than the rest. It can be said that the IGT is the best performing algorithm due to its small RPDs and fast computation time. It is worth to mention that all proposed algorithms in this paper are able to find the current best-known C_{max} values reported in Cui and Gu (2015), as shown in Table 9. Furthermore, three of the best-known results are improved by the proposed algorithms, which are shown in bold. Thus, the BC_{max} column of Table 9 reports the updated best-known C_{max} values to the literature.

For large instances (with 40, 50 and 60 jobs) in Table 10, the optimal results cannot be obtained by CPLEX within the time limit, as expected; CPLEX cannot find any of the BC_{max} values for these

Table 10
Comparison on instances with 40, 50 and 60 jobs based on C_{max} criterion.

Instance	IGT_{ALL}				IGT				VBIH				DABC				IDABC				CPLEX		BC_{MAX}
	RPD(%)			Avg. CPU (s)	RPD(%)			Avg. CPU (s)	RPD(%)			Avg. CPU (s)	RPD(%)			Avg. CPU (s)	RPD(%)			Avg. CPU (s)	C_{MAX}	RPD(%)	
	Avg	Min	Max		Avg	Min	Max		Avg	Min	Max		Avg	Min	Max		Avg	Min	Max				
j40c5e1	0.07	0.00	0.15	9.04	0.08	0.00	0.15	10.91	0.07	0.00	0.15	9.20	0.15	0.15	0.15	1.05	0.15	0.15	0.15	2.69	738	7.27	688
j40c5e2	0.01	0.00	0.13	15.07	0.00	0.00	0.00	5.49	0.00	0.00	0.00	11.86	0.02	0.00	0.39	9.42	0.11	0.00	0.26	28.91	850	10.82	767
j40c5e3	0.00	0.00	0.00	2.32	0.00	0.00	0.00	2.35	0.00	0.00	0.00	4.62	0.04	0.00	0.25	4.96	0.09	0.00	0.25	15.26	880	9.86	801
j40c5e4	0.14	0.14	0.27	18.40	0.07	0.00	0.14	14.79	0.14	0.00	0.27	11.45	0.10	0.00	0.14	13.98	0.23	0.00	0.68	31.38	863	17.57	734
j40c5e5	0.00	0.00	0.00	1.73	0.00	0.00	0.00	0.73	0.00	0.00	0.00	1.78	0.01	0.00	0.14	2.25	0.00	0.00	0.00	14.88	829	16.27	713
j40c5e6	0.00	0.00	0.00	0.49	0.00	0.00	0.00	0.31	0.00	0.00	0.00	1.07	0.00	0.00	0.00	0.78	0.00	0.00	0.00	2.93	859	9.99	781
j40c5e7	0.12	0.00	0.14	2.32	0.12	0.00	0.14	3.04	0.13	0.00	0.14	3.16	0.10	0.00	0.14	6.46	0.14	0.14	0.14	5.01	856	16.46	735
j40c5e8	0.08	0.00	0.12	17.59	0.05	0.00	0.12	14.65	0.14	0.00	0.25	13.41	0.21	0.12	0.25	2.19	0.22	0.00	0.25	6.58	883	10.10	802
j40c5e9	0.00	0.00	0.00	2.87	0.00	0.00	0.00	1.19	0.00	0.00	0.00	3.16	0.00	0.00	0.00	3.10	0.01	0.00	0.12	16.09	915	8.93	840
j40c5e10	0.28	0.00	0.39	20.35	0.19	0.13	0.26	18.05	0.23	0.13	0.39	16.82	0.14	0.13	0.26	20.50	0.39	0.13	0.65	27.71	966	24.81	774
j50c5e1	0.08	0.00	0.21	22.32	0.05	0.00	0.21	13.68	0.04	0.00	0.21	24.38	0.19	0.00	0.31	12.61	0.23	0.00	0.41	20.53	1332	37.32	970
j50c5e2	0.00	0.00	0.00	5.55	0.00	0.00	0.00	7.91	0.00	0.00	0.00	11.06	0.15	0.00	0.70	11.30	0.09	0.00	0.20	17.30	1248	25.55	994
j50c5e3	0.20	0.10	0.29	16.63	0.13	0.00	0.29	21.54	0.22	0.10	0.29	14.04	0.52	0.10	1.07	21.01	0.45	0.10	1.07	30.88	1241	20.60	1029
j50c5e4	0.00	0.00	0.00	2.74	0.00	0.00	0.00	1.81	0.00	0.00	0.00	2.60	0.00	0.00	0.00	2.93	0.06	0.00	0.33	24.16	1291	42.02	909
j50c5e5	0.12	0.00	0.19	22.76	0.03	0.00	0.10	22.06	0.04	0.00	0.19	24.42	0.06	0.00	0.39	26.82	0.35	0.10	0.68	33.22	1196	16.57	1026
j50c5e6	0.27	0.12	0.36	14.22	0.22	0.00	0.36	21.75	0.29	0.00	0.48	13.85	0.33	0.00	0.48	13.42	0.40	0.12	0.71	19.16	1019	21.31	840
j50c5e7	0.06	0.00	0.11	8.94	0.03	0.00	0.11	15.84	0.04	0.00	0.11	13.93	0.25	0.00	0.65	7.97	0.15	0.11	0.54	17.74	1135	22.31	928
j50c5e8	0.00	0.00	0.00	3.98	0.00	0.00	0.00	1.93	0.00	0.00	0.00	3.17	0.01	0.00	0.12	5.39	0.11	0.00	0.35	20.77	998	16.45	857
j50c5e9	0.29	0.00	0.61	26.27	0.25	0.15	0.46	24.37	0.25	0.15	0.46	24.65	0.22	0.00	0.61	24.75	0.85	0.31	1.53	34.99	831	27.26	653
j50c5e10	0.00	0.00	0.00	7.63	0.00	0.00	0.00	4.83	0.00	0.00	0.00	10.71	0.00	0.00	0.00	11.10	0.21	0.00	0.52	24.27	1117	15.27	969
j60c5e1	0.04	0.00	0.09	14.92	0.00	0.00	0.09	18.44	0.06	0.00	0.09	14.83	0.00	0.00	0.09	7.41	0.09	0.00	0.18	20.23	1295	14.00	1136
j60c5e2	0.54	0.22	0.76	26.37	0.25	0.11	0.54	28.86	0.32	0.00	0.65	23.77	0.22	0.00	0.54	27.29	1.05	0.65	1.52	41.85	1380	50.00	920
j60c5e3	0.00	0.00	0.00	2.44	0.00	0.00	0.00	1.75	0.00	0.00	0.00	3.83	0.02	0.00	0.43	5.93	0.06	0.00	0.43	31.12	1353	16.84	1158
j60c5e4	0.42	0.25	0.63	24.48	0.23	0.00	0.38	29.59	0.32	0.13	0.51	27.34	0.20	0.00	0.76	32.85	0.91	0.25	1.65	46.58	1116	41.62	788
j60c5e5	0.04	0.00	0.19	22.62	0.01	0.00	0.09	10.31	0.02	0.00	0.09	20.74	0.04	0.00	0.38	10.22	0.37	0.00	0.66	33.79	1503	41.53	1062
j60c5e6	0.22	0.10	0.38	20.04	0.12	0.00	0.19	20.79	0.19	0.00	0.29	19.48	0.20	0.10	0.29	15.26	0.46	0.19	0.67	30.42	1385	32.16	1048
j60c5e7	0.00	0.00	0.00	4.46	0.00	0.00	0.00	3.40	0.00	0.00	0.00	12.02	0.06	0.00	0.24	4.09	0.02	0.00	0.24	13.63	1432	16.90	1225
j60c5e8	0.00	0.00	0.00	5.59	0.00	0.00	0.00	10.69	0.01	0.00	0.10	20.76	0.05	0.00	0.10	3.17	0.00	0.00	0.00	7.17	1359	31.56	1033
j60c5e9	0.00	0.00	0.00	1.91	0.00	0.00	0.00	2.11	0.00	0.00	0.00	13.83	0.05	0.00	0.26	3.95	0.00	0.00	0.00	5.83	1377	19.12	1156
j60c5e10	0.20	0.00	0.22	9.99	0.19	0.00	0.22	9.83	0.22	0.22	0.22	7.80	0.41	0.22	1.09	9.85	0.46	0.22	0.65	25.82	1253	36.05	921
Average	**0.11**	**0.03**	**0.17**	**11.80**	**0.07**	**0.01**	**0.13**	**11.43**	**0.09**	**0.02**	**0.16**	**12.79**	**0.13**	**0.03**	**0.34**	**10.73**	**0.25**	**0.08**	**0.49**	**21.70**		**22.55**	

Table 11
Wilcoxon signed-rank test results for large instances based on C_{max} criterion.

Pairs of algorithms	p-value	Significant? ($p \leq 0.05$)	Decision
IGT_{ALL} vs IGT	0.000	Yes	Reject H_0
IGT_{ALL} vs VBIH	0.151	No	Fail to reject H_0
IGT_{ALL} vs DABC	0.253	No	Fail to reject H_0
IGT_{ALL} vs IDABC	0.000	Yes	Reject H_0
IGT vs VBIH	0.001	Yes	Reject H_0
IGT vs DABC	0.001	Yes	Reject H_0
IGT vs IDABC	0.000	Yes	Reject H_0
VBIH vs DABC	0.064	No	Fail to reject H_0
VBIH vs IDABC	0.000	Yes	Reject H_0
DABC vs IDABC	0.004	Yes	Reject H_0

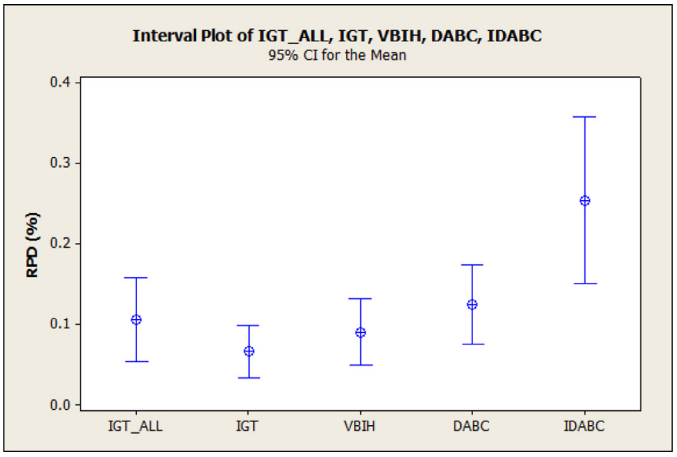

Fig. 17. Interval plot of algorithms based on C_{max} criterion.

30 instances and its average RPD is 22.55%. All proposed heuristic algorithms clearly outperform time-limited CPLEX for these instances.

In terms of the number of BC_{max} values found, IGT_{ALL} finds 24, IGT finds 27, VBIH finds 25, DABC finds 24, and IDABC finds 18 of the BC_{max} values out of 30 instances. In terms of average RPDs, IGT (0.07%), VBIH (0.09%), IGT_{ALL} (0.11%) and DABC (0.13%) perform much better than IDABC (0.25%). Furthermore, in terms of maximum RPDs, IGT (0.13%), VBIH (0.16%) and IGT_{ALL} (0.17%) perform significantly better than DABC (0.34%) and IDABC (0.49%). For the average computation times, IGT, VBIH, IGT_{ALL} and DABC are faster than IDABC, as their average CPU times are between 10 and 13 s, and DABC is slightly faster than all others. Consequently, it can be said that IGT and VBIH perform slightly better than others due to their smaller RPDs.

To test performance of the algorithms statistically, we conducted a series of Wilcoxon signed-rank tests at the significance level $\alpha = 0.05$, using the same hypothesis testing methodology in Section 6.2. Note that, the sample size is 30 for each pairwise comparison of algorithms. The Wilcoxon signed-rank test results for the pairwise comparisons of algorithms are reported in Table 11. As shown in the table, all the pairwise differences are statistically significant at the $\alpha = 0.05$ level, except IGT_{ALL} vs VBIH, IGT_{ALL} vs DABC and VBIH vs DABC pairs. For these three pairs of algorithms, the differences between the algorithms are not meaningful at the $\alpha = 0.05$ level, implying that these three algorithms show similar performances.

In order to justify the above statements, we also provide the interval plots for these algorithms in Fig. 17. The plot shows that IGT, VBIH, IGT_{ALL} and DABC are statistically equivalent, as their confidence intervals overlap. However, IGT and VBIH are statistically better than IDABC, as their confidence intervals do not coincide.

7. Conclusion

In this study, we address the well-known HFSP which has many real-life applications in industry. We consider two important objectives for this problem as the makespan and the total flow time. Even though many solution approaches have been presented for the HFSP with makespan criterion, the current literature on HFSP with total flow time criterion is rather thin. We propose a mathematical model, four variants of IG heuristic (IG_{RS}, IG_{GR}, IGT and IGT_{ALL}) and a VBIH for the HFSP with total flow time criterion. We calibrate the parameters of the proposed heuristics through detailed experimentation and test the performances of constructive heuristics. The results show that the proposed GRASP_NEH(x) is a very effective constructive heuristic for the HFSP.

In the computational study, the well-known HFSP benchmark suite (Carlier and Neron, 2000; Liao et al., 2012), which includes 87 instances, is used to evaluate the performances of the proposed solution approaches. As there are only 10 large instances in this suite, we generate additional 30 large instances and propose these to the literature as new benchmark problems. CPLEX results under a time limit are obtained for all of these 117 instances by solving the mathematical model. The optimal total flow time results are obtained for 26 instances of Carlier and Neron (2000), and gaps from optimality are reported for any instance that cannot be solved to optimality.

The performances of proposed heuristics are also evaluated using the time-limited CPLEX results. The algorithms provide remarkable results for the benchmarks by improving most non-optimal CPLEX solutions. The proposed algorithms are also compared with DABC (Pan et al., 2014) and IDABC (Cui and Gu, 2015). The results show that the proposed IGT, IGT_{ALL} and VBIH algorithms outperform other solution methods in terms of both solution quality and computation time. Especially for large instances, IGT_{ALL} and VBIH perform much better than IG_{RS}, IG_{GR}, DABC and IDABC.

Furthermore, the proposed algorithms are tested on large instances for the makespan criterion. For the instances with 30 jobs (Liao et al., 2012), the results show that the proposed algorithms are able to find the current best-known C_{max} values reported in Cui and Gu (2015), and improve three best-known solutions. For other large instances, the proposed algorithms are compared with DABC (Pan et al., 2014) and IDABC (Cui and Gu, 2015), which are the two best performing algorithms for the HFSP with makespan minimization. The results indicate that the proposed heuristics also perform superbly for the makespan criterion, while IGT and VBIH perform slightly better than the rest in terms of solution quality.

Through the establishment of the new benchmark instances, we expect that our study will encourage researchers to work on HFSP with total flow time criterion, which is a rarely studied variant in the current literature. For future studies, different metaheuristics can be developed for the problem and compared with the existing results. Additionally, the proposed algorithms in this study can be applied to more complex and realistic scheduling problems, such as those with setup times, release dates, and no-wait or blocking limitations. Different performance measures such as tardiness can also be handled by the inclusion of due dates for the proposed problem instances.

Acknowledgments

M. Fatih Tasgetiren and Quan-Ke Pan acknowledge the HUST Project in Wuhan in China. They are supported by the National Natural Science Foundation of China (grant no: 51435009). The authors would also like to thank the anonymous referees for their insightful comments on a previous version of the paper.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cor.2019.06.009.

References

Johnson, S.M., 1954. Optimal two-and three-stage production schedules with setup times included. Naval Res. Logistics Quart. 1 (1), 61–68.

Ribas, I., Leisten, R., Framinan, J.M., 2010. Review and classification of hybrid flowshop scheduling problems from a production system and a solutions procedure perspective. Comput. Oper. Res. 37 (8), 1439–1454.

Pan, Q., Wang, L., Mao, K., Zhao, J.H., Zhang, M., 2013. An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steel making process. IEEE Trans. Autom. Sci. Eng. 10 (2), 307–322.

Sherali, H.D., Sarin, S.C., Kodialam, M.S., 1990. Models and algorithms for a two-stage production process. Prod. Plan. Control 1 (1), 27–39.

Grabowski, J., Pempera, J., 2000. Sequencing of jobs in some production system. Eur. J. Oper. Res. 125 (3), 535–550.

Wittrock, J.R., 1988. An adaptable scheduling algorithm for flexible flow lines. Oper. Res. 36 (3), 445–453.

Liu, C.Y., Chang, S.C., 2000. Scheduling flexible flowshops with sequence dependent setup effects. IEEE Trans. Robot. Autom. 16 (4), 408–419.

Jin, Z.H., Ohno, K., Ito, T., Elmaghraby, S.E., 2002. Scheduling hybrid flowshops in printed circuit board assembly lines. Prod. Oper. Manage. 11 (2), 216–230.

Framinan, J.M., Leisten, R., 2003. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. Omega (Westport) 31 (4), 311–317.

Carlier, J., Neron, E., 2000. An exact method for solving the multiprocessor flowshop. R.A.I.R.O. Oper. Res.. 34, 1–25.

Liao, C.J., Tjandradjaj, E., Chung, T.P., 2012. An approach using particle swarm optimization and bottleneck heuristic to solve flowshop scheduling problem. Appl. Soft. Comput. 12 (6), 1755–1764.

Ruiz, R., Vazquez Rodriguez, J.A., 2010. The hybrid flowshop scheduling problem. Eur. J. Oper. Res. 205, 1–18.

Brah, S.A., Hunsucker, J.L., 1991. Branch and bound algorithm for the flow shop with multiple processors. Eur. J. Oper. Res. 51 (1), 88–99.

Neron, E., Baptiste, P., Gupta, J.N.D., 2001. Solving hybrid flow shop problem using energetic reasoning and global operations. Omega (Westport) 29 (6), 501–511.

Rajendran, C., Chaudhuri, D., 1992a. Scheduling in n-jobs m stage flowshop with parallel processors to minimize makespan. Int. J. Prod. Econ. 27 (2), 137–143.

Moursli, O., Pochet, Y., 2000. A branch-and-bound algorithm for the hybrid flowshop. Int. J. Prod. Econ. 64 (1–3), 113–125.

Portmann, M.C., Vignier, A., Dardilhac, D., Dezalay, D., 1998. Branch and bound crossed with GA to solve hybrid flowshops. Eur. J.Oper. Res. 107 (2), 389–400.

Morita, H., Shio, N., 2005. Hybrid branch and bound method with genetic algorithm for flexible flowshop scheduling problem. JSME Int. J. Series C 48 (1), 46–52.

Rajendran, C., Chaudhuri, D., 1992b. A multistage parallel-processor flowshop problem with minimum flowtime. Eur. J. Oper. Res. 57 (1), 111–122.

Vignier, A., Dardilhac, D., Dezalay, D., Proust, C., 1996. A branch and bound approach to minimize the total completion time in a k-stage hybrid flowshop. In: Proceedings of the Fifth International Conference on Emerging Technologies and Factory Automation (ETFA'96). IEEE Press, pp. 215–220.

Azizoğlu, M., Çakmak, E., Kondakci, S., 2001. A flexible flowshop problem with total flow time minimization. Eur. J. Oper. Res. 132 (3), 528–538.

Tang, L.X., Xuan, H., Liu, J.Y., 2006. A new lagrangian relaxation algorithm for hybrid flowshop scheduling to minimize total weighted completion time. Comput. Oper. Res. 33 (11), 3344–3359.

Kis, T., Pesch, E., 2005. A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. Eur. J. Oper. Res 164 (3), 592–608.

Gupta, J.N.D., 1988. Two-stage, hybrid flow shop scheduling problem. J. Oper. Res. Soc. 39 (4), 359–364.

Haouari, M., M'Hallah, R., 1997. Heuristic algorithms for the two-stage hybrid flowshop problem. Oper. Res. Lett. 21 (1), 43–53.

Nowicki, E., Smutnick, C., 1998. The flow shop with parallel machines: a tabu search approach. Eur. J. Oper. Res. 106 (2–3), 226–253.

Negenman, E.G., 2001. Local search algorithms for the multiprocessor flow shop scheduling problem. Eur. J. Oper. Res. 128 (1), 147–158.

Alaykyran, K., Engin, O., Doyen, A., 2007. Using ant colony optimization to solve hybrid flow shop scheduling problems. Int. J. Adv. Manuf. Technol. 35 (5–6), 541–550.

Kahraman, C., Engin, O., Kaya, I., Yilmaz, M.K., 2008. An application of effective genetic algorithms for solving hybrid flow shop scheduling problems. Int. J. Comput. Intell. Syst. 1 (2), 134–147.

Xiao, W., Hao, P., Zhang, S., Xu, X., 2000. Hybrid flow shop scheduling using genetic algorithms. In: Proceedings of the Third World Congress on Intelligent Control and Automation. IEEE Press, pp. 537–541.

Jin, Z., Yang, Z., Ito, T., 2006. Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. Int. J. Prod. Econ. 100, 322–334.

Engin, O., Doyen, A., 2004. A new approach to solve hybrid flow shop scheduling problems by artificial immune system. Future Gener. Comput. Syst. 20 (6), 1083–1095.

Niu, Q., Zhou, T., Ma, S., 2009. A quantum-inspired immune algorithm for hybrid flow shop with makespan criterion. J. Univ. Comp. Sci. 15, 765–785.

Pan, Q., Wang, L., Li, J.Q., Duan, J.H., 2014. A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. Omega (Westport) 45, 42–56.

Cui, Z., Gu, X., 2015. An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems. Neurocomputing 148, 248–259.

Kizilay, D., Tasgetiren, M.F., Pan, Q., Wang, L., 2014. An iterated greedy algorithm for the hybrid flowshop problem with makespan criterion. In: Computational Intelligence in Production and Logistics Systems (CIPLS). IEEE Symposium, pp. 16–23. on.

Brah, S., Loo, L.L., 1999. Heuristics for scheduling in a flow shop with multiple processors. Eur. J. Oper. Res. 113 (1), 113–122.

Allahverdi, A., Al-Anzi, F.S., 2006. Scheduling multi-stage parallel-processor services to minimize average response time. J. Oper. Res. Soc. 57 (1), 101–110.

Niu, Q., Zhou, T., Fei, M., Wang, B., 2012. An efficient quantum immune algorithm to minimize mean flow time for hybrid flow shop problems. Math Comput. Simul. 84, 1–25.

Pan, Q., Dong, Y., 2014. An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimization. Inf. Sci. (NY) 277, 643–655.

Low, C.Y., 2005. Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. Comput. Oper. Res. 32 (8), 2013–2025.

Naderi, B., Zandieh, M., Balagh, A.K.G., Roshanaei, V., 2009. An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion and total tardiness. Expert Syst. Appl. 36 (6), 9625–9633.

Öztop, H., Tasgetiren, M.F., Eliiyi, D.T., Pan, Q., 2018. Iterated greedy algorithms for the hybrid flowshop scheduling with total flow time minimization. In: Proceedings of ACM Genetic and Evolutionary Computation Conference (GECCO'18), pp. 379–385. 2018 https://doi.org/10.1145/3205455.3205500 .

Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur. J. Oper. Res. 177 (3), 2033–2049.

Vignier, A., Billaut, J.C., Proust, C., 1999. Hybrid flowshop scheduling problems: state of the art. Rairo - Recherche Operationnelle – Operations Res. 33 (2), 117–183.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan Rinnooy, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Anna. Discrete Math. 5, 287–326.

Nawaz Jr, M., Enscore, E.E., Ham, I., 1983. A heuristic algorithm for the m-machine n-job flowshop sequencing problem. Omega (Westport) 11 (1), 91–95.

Feo, T., Resende, M.G.C., 1995. Greedy randomizes adaptive search procedures. J. Global Opt. 6 (2), 109–133.

Osman, I., Potts, C., 1989. Simulated annealing for permutation flow-shop scheduling. Omega (Westport) 17 (6), 551–557.

Tasgetiren, M.F., Kizilay, D., Pan, Q., Suganthan, P.N., 2017. Iterated greedy algorithms for the blocking flowshop scheduling problem with the makespan criterion. Comput. Operations Res. 77, 111–126.

Pan, Q., Tasgetiren, M.F., Liang, Y.C., 2008. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Comput. Ind. Eng. 55 (4), 795–816.

Tasgetiren, M.F., Pan, Q., Kizilay, D., Gao, K., 2016a. A variable block insertion heuristic for the blocking flowshop scheduling problem with total flowtime criterion. Algorithms 9 (4), 71. doi:10.3390/a9040071.

Dubois-Lacoste, J., Pagnozzi, F., Stützle, T., 2017. An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. Comput. Oper. Res. 81, 160–166.

Subramanian, A., Battarra, M., Potts, C., 2014. An iterated local search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. Int. J. Prod. Res. 52 (9), 2729–2742.

Xu, H., Lu, Z., Cheng, T., 2014. Iterated local search for single-machine scheduling with sequence-dependent setup times to minimize total weighted tardiness. J. Schedul. 17 (3), 271–287.

Tasgetiren, M.F., Pan, Q., Ozturkoglu, Y., Chen, A.H.L., 2016b. A memetic algorithm with a variable block insertion heuristic for single machine total weighted tardiness problem with sequence dependent setup times. IEEE Congress Evol. Comput. CEC 2911–2918 2016.

Tasgetiren, M.F., Pan, Q., Kizilay, D., Vélez-Gallego, M.C., 2017. A variable block insertion heuristic for permutation flowshops with makespan criterion. IEEE Congress Evol.y Comput. CEC 12 (5), 726–733 2017.

Montgomery, D.C., 2008. Design and Analysis of Experiments. John Wiley & Sons 2008.