Engineering Applications of Artificial Intelligence 65 (2017) 178–199

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

# A hybrid harmony search algorithm with efficient job sequence scheme and variable neighborhood search for the permutation flow shop scheduling problems

Fuqing Zhao [a,*], Yang Liu [a], Yi Zhang [b], Weimin Ma [c], Chuck Zhang [d]

[a] School of Computer and Communication Technology, Lanzhou University of Technology, Lanzhou 730050, China
[b] School of Mechanical Engineering, Xijing University, Xi'an 710123, China
[c] School of Economics and Management, Tongji University, Shanghai 200092, China
[d] H. Milton Stewart School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA

ARTICLE INFO

*Keywords:*
Permutation flow shop scheduling
Evolution computation
Optimization
Operation research
Harmony search
Parameter sensitivity

ABSTRACT

The permutation flow shop scheduling problem (PFSSP), one of the most widely studied production scheduling problems, is a typical NP-hard combinatorial optimization problem. In this paper, a hybrid harmony search algorithm with efficient job sequence mapping scheme and variable neighborhood search (VNS), named HHS, is proposed to solve the PFFSP with the objective to minimize the makespan. First of all, to extend the HHS algorithm to solve the PFSSP effectively, an efficient smallest order value (SOV) rule based on random key is introduced to convert continuous harmony vector into a discrete job permutation after fully investigating the effect of different job sequence mapping schemes. Secondly, an effective initialization scheme, which is based on NEH heuristic mechanism combining with chaotic sequence, is employed with the aim of improving the solution's quality of the initial harmony memory (HM). Thirdly, an opposition-based learning technique in the selection process and the best harmony (best individual) in the pitch adjustment process are made full use of to accelerate convergence performances and improve solution accuracy. Meanwhile, the parameter sensitivity is studied to investigate the properties of HHS, and the recommended values of parameters adopted in HHS are presented. Finally, by making use of a novel variable neighborhood search, the efficient insert and swap structures are incorporated into the HHS to adequately emphasize local exploitation ability. Experimental simulations and comparisons on both continuous and combinatorial benchmark problems demonstrate that the HHS algorithm outperforms the standard HS algorithm and other recently proposed efficient algorithms in terms of solution quality and stability.



## 1. Introduction

The production scheduling problem is a decision-making process which plays a crucial role in the manufacturing system, and widely exists in the real-world engineering fields (Pinedo, 2012). The permutation flow shop scheduling problem (PFSSP, all the expansions for the abbreviations using in this paper are listed in Table 19 in "Appendix A"), as one of the classical flow shop scheduling problems (FSSP), can be regarded as a simplified version of FSSP. The PFSSP, with the objective of minimizing the maximum completion time (namely makespan), is well-studied and has received an enormous amount of attention. As for the computational complexity, it has been proved that PFSSP with more than two machines is NP-hard (Blazewicz et al., 1983; Rand, 1977). Therefore, it is of significance both in theory and engineering applications to develop effective and efficient approaches for PFSSP.

The PFSSP was first researched by Johnson (Johnson, 1954) on the two machines, and since then a large amount of optimization approaches have been proposed for solving the PFSSP with criterion of minimizing makespan. Due to the complexity of PFSSP, many efforts have been devoted to searching for high-quality solutions in an acceptable computational time. Approaches for PFSSP can be classified into two categories: exact algorithms and heuristic algorithms. For the exact algorithms, such as branch-and-bound method, dynamical programming

* Corresponding author.
*E-mail addresses:* Fzhao2000@hotmail.com (F. Zhao), 2476938557@qq.com (Y. Liu), 1049440546@qq.com (Y. Zhang), mawm@tongji.edu.cn (W. Ma), chuck.zhang@isye.gatech.edu (C. Zhang).

http://dx.doi.org/10.1016/j.engappai.2017.07.023
Received 31 October 2016; Received in revised form 19 July 2017; Accepted 26 July 2017
Available online 23 August 2017
0952-1976/

and linear programming can be only applied in solving the instances with small sizes (Della Croce et al., 1996; Ignall and Schrage, 1965; Stafford, 1988). However, for the large-scale instances, the results obtained by exact algorithms in terms of the solution quality in a reasonable time are not satisfactory. Therefore, the heuristic algorithms have been proposed.

Heuristics can be divided into three categories: constructive heuristic algorithms, improvement heuristic algorithms and hybrid heuristic algorithms. The constructive algorithms (Nawaz et al., 1983; Rajendran, 1993) are simple and fast. It can build a feasible scheduling solution through the constructive operations and usually nearly optimal solutions can be acquired in an acceptable time, while the solutions in terms of the quality are not ideal. For the improvement heuristic algorithms, most of them are the meta-heuristics. Classical meta-heuristics include Genetic algorithm (Reeves, 1995), Particle swarm optimization algorithm (Eberhart and Kennedy, 1995; Liu et al., 2007; Zhao et al., 2014b), Shuffled complex evolution algorithm (Zhao et al., 2014c, 2015c), Tabu search algorithm (Grabowski and Wodecki, 2004; Nowicki and Smutnicki, 1996), Simulated annealing (Laha and Chakraborty, 2010; Osman and Potts, 1989), VNS (Hansen and Mladenović, 2001; Moslehi and Khorasanian, 2014), Differential evolution algorithm (Li and Yin, 2013b; Storn and Price, 1997), Estimation of distribution algorithm (Wang et al., 2013; Zhao et al., 2015b), Bacterial foraging algorithm (Zhao et al., 2014a), Iterated greedy algorithm (Ruiz and Stützle, 2007) and Cuckoo search algorithm (Li and Yin, 2013a). More pleasing solutions can be obtained by using the improvement heuristic algorithm, but the evolutionary process is often time-consuming.

It is clear that a single meta-heuristic algorithm for the PFSSP has many limitations. Therefore, in order to improve the performance of the meta-heuristic, combination of two or more meta-heuristics, namely hybridization has become a hot topic and attracted much attention. The idea of hybridization is drawing strong points of the other meta-heuristics to offset itself weakness. This kind of hybridization can generate a pleasing solution in a reasonable computational time especially for large-scale instance.

In the past decades, researchers focus on the hybridization to find high-quality solutions in a reasonable time and a lot of approaches for PFSSP have been proposed. To find the minimum makespan of permutation flow shop sequencing problem, a genetic algorithm was developed by Reeves (1995). Zheng and Wang (2003) proposed an effective heuristic for flow shop scheduling. In the proposed algorithm, the NEH heuristic was employed for the population initialization and a metropolis sample of simulated annealing with probabilistic jump was incorporated into GA to strengthen the ability of local search. To solve the sequence-independent permutation flow-shop scheduling problem, a novel hybrid genetic algorithm (HGA) was introduced by Mirabi (2014). The proposed HGA made use of a modified methodology to initialize the population and an iterative swap heuristic was utilized to improve them. By utilizing the smallest position value (SPV) which is used to convert continuous vector into a discrete job permutation, A PSO algorithm for the PFSSP with objectives of minimizing makespan and total flow time criterions, namely PSOVNS, was proposed by Tasgetiren et al. (2007). The effective heuristic called variable neighborhood search (VNS) was used as local search to further enhance the ability of obtaining global solutions. Based on the memetic algorithm, an effective PSO algorithm (PSOMA) for PFSSP was developed by Liu et al. (2007), which used a simulated annealing based local search with several different neighborhood operators to balance the exploration and exploitation. Zhang and Sun (2009) applied PSO algorithm to solve the PFSSP with makespan criterion by using an alternate two phases. In the proposed algorithm, two evolutionary processes were executed alternatively to speed up the convergence performance. Based on a tabu search method with specific neighborhood definitions, a fast algorithm for the problem of finding the minimizing makespan in the PFSSP was presented by Nowicki and Smutnicki (1996), where the block of jobs was introduced. In (Gao et al., 2013), a new tabu search algorithm that combined an enhanced local search for solving the distributed permutation flow shop scheduling problem was exploited. In order to obtain the stable performance, two simulated annealing algorithm with a modified generation mechanisms were presented by Ishibuchi et al. (1995). Liu proposed a hybrid variable neighborhood search (HVNS) combining with the simulated annealing algorithm to solve the limited-buffer permutation flow shop scheduling problem with objective of minimizing makspan. Qian et al. (2008) presented a hybrid differential evolution algorithm for the single-objective PFSSP, called HDE, which designed a simple but efficient local search according to the landscape of PFSSP to reinforce the exploitation. Liu et al. (2014) applied a hybrid DE algorithm to deal with the PFSSP, which combined the individual improving scheme (IIS) and Greedy-based local search to enhance the solution quality. A hybrid cuckoo search (HCS) algorithm with objectives of minimizing makespan and total flow time for solving PFSSP was developed by Li and Yin (2013a). In the HCS, VNS algorithm with a certain probability was adopted to emphasize exploitation. In brief, different kinds of hybridizations (Dong et al., 2015; Liu et al., 2011; Xie et al., 2014) emerge in endlessly. Literatures (Baskar, 2015; Pan and Ruiz, 2013; Ruiz and Maroto, 2005) make a detailed survey about the PFSSP issue with different hybridized heuristics, which are good references.

Harmony search (HS) algorithm, as a new meta-heuristic, was proposed by Geem et al. (2001), which is one of the effective approaches for addressing combinational optimization problems. The basic idea of HS algorithm comes from the music improvisation, and mimics the process that musicians repeatedly adjust the pitches of different instruments to reach a pleasing harmony eventually. Compared with the earlier meta-heuristics, HS algorithm has many advantages such as simple concept, few parameters to be tuned and easy to be implemented, and also owns a particular behavior of exploring and exploiting the search space, which has made it quite successful in the real world (Del Ser et al., 2012; Geem, 2007; Geem et al., 2005; Xiang et al., 2014; Zhao et al., 2015a). Therefore, HS is selected as the main algorithm in this paper. For the scheduling problems, many approaches based on HS have been proposed. A chaotic harmony search (CHS) was presented by Pan et al. (2011b) to minimize the makespan for PFSSP with limited buffers. In the CHS, NEH heuristic was employed for the population initialization and a chaotic local search with probabilistic jumping scheme was designed to adequately perform exploitation. Wang et al. (2010) applied the hybrid harmony search algorithm to solve the flow shop with blocking. In order to deal with the no-wait flow shop scheduling problem with makespan criterion, a discrete HS algorithm was proposed by Gao et al. (2011). Pan et al. (2011a) proposed a local-best harmony search algorithm to handle lot-streaming flow shop scheduling problem with objectives to minimize the total weighted earliness and tardiness penalties. Yuan et al. (2013) developed a hybrid HS algorithm for the flexible job shop scheduling problem (FJSP) with the criterion to minimize makespan, and in the proposed algorithm, an improved neighborhood structure based on the critical path was adopted to speed up the local search. In Gao et al. (2014), an effective discrete harmony search (DHS) algorithm to solve flexible job shop scheduling problem (FJSP) with multiple objectives was proposed, which adopted several local search methodologies to local exploitation capability. Li et al. (2015) presented a hybrid HS algorithm to solve the multi-objective flow line manufacturing cell scheduling problem, which adopted crossover operator for diversification and employed iterative local search to further improve the solution quality.

However, from the aforementioned introduction for HS algorithm, the research on the HS for sequencing problem is still considerably limited because of its continuous nature. In this paper, a hybrid harmony search algorithm with efficient job sequence scheme and variable neighborhood search (VNS), named HHS, is proposed to solve the PFSSP with the objective to minimize the makespan. To make HHS algorithm suitable for solving PFSSP effectively, an efficient smallest order value (SOV) rule is introduced to address the conversion problem between the continuous harmony vector and the job permutation. Meanwhile, NEH

heuristic and chaotic sequence is used to initialize the harmony memory and improve the initial solution quality. Then the best individual in HM is extracted to speed up the population convergence performance. And the parameter sensitivity is also analyzed. Besides, an efficient insert and swap structures based on a novel variable neighborhood search are embedded into HHS to stress the local search ability. The statistical results on both continuous and combinatorial benchmark problems reveal that the HHS algorithm is more efficient than the standard HS algorithm and other recently proposed algorithms in terms of solution quality and stability.

The remainder of the paper is organized as follows. The description of PFSSP is provided in Section 2. The standard HS algorithm and the implement details of HHS for PFSSP are described in Section 3. The experimental results and comparisons, along with their analysis and discussions are stated in Section 4. Finally, we end the paper with some conclusions and future work in Section 5.

## 2. Description of the permutation flow shop scheduling problem

The PFSSP is made of a set of $n$ jobs on $s$ set of $m$ machines. In the PFSSP, $n$ jobs $j = j_1, j_2, \dots, j_n$, are sequentially processed on set of $m$ machines $M = \{M_1, M_1, \dots, M_m\}$. Thus, each job should be provided an operation $O_j = O_{j1}, O_{j2}, \dots, O_{jm}$. Each job has a given processing time represented by $P_{i,j} = (i = 1, 2, \dots, n, j = 1, 2, \dots, m)$. At any moment, each job can be processed at most on one machine, and each machine can process at most one job. Besides, the operations cannot be interrupted once the first job on the first machine has launched. The sequences of the jobs to be processed are the same on each machine. Therefore, each instance has $n!$ possible processing sequences in total. The goal is to minimize completion time (makespan), which is the most commonly studied objective of PFSSP.

With the objective to minimize the makespan, the PFSSP can be denoted as $n/m/P/C_{max}$, where $n$ represents the number of jobs and $m$ stands for machines. $P$ denotes permutation flow shop scheduling problem. $C_{max}$ is the objective to minimize the completion time. Let $\pi = \{\pi_1, \pi_2, \dots, \pi_n\}$ stand for a job permutation, where $\pi$ is the set of all the permutations of the $n$ jobs. Let $C(\pi_i, m)$ be the completion time of jobs $\pi_i$ on the machine $m$. Let $p_{\pi_i,j}$ denote the processing time of job $\pi_i$ on the machine $j$. The mathematical formulation of the PFSSP can be described as follows:

$$\begin{aligned} C(\pi_1, 1) &= p_{\pi_1,1} \\ C(\pi_i, 1) &= C(\pi_{i-1}, 1) + p_{\pi_1,1},\ i = 2, \dots, n \\ C(\pi_1, j) &= C(\pi_1, j-1) + p_{\pi_1,j},\ j = 2, \dots, m \\ C(\pi_i, j) &= \max\left(C(\pi_{i-1}, j), C(\pi_i, j-1)\right) + p_{\pi_i,j}, \\ &\quad i = 2, \dots, n,\ j = 2, \dots, m. \end{aligned}$$

Usually, the makespan of a permutation $\pi$ is the completion time of the last job on the last machine; therefore, the makespan can be defined as:

$$C_{\max}(\pi) = C(\pi_n, m).$$

The objective of PFSSP is to find the optimal permutation $\pi^*$ in the set of permutations:

$$C(\pi^*) \le C(\pi_n, m), \quad \forall \pi \in \prod.$$

## 3. The hybrid harmony search (HHS) for PFSSP

### 3.1. The standard harmony search algorithm

Harmony search algorithm is one of the latest evolutionary computation meta-heuristics. Musical performances seek a best state (fantastic harmony) determined by aesthetic evaluation, which is a set of the sounds played by joint instruments, as the meta-heuristics seek a best state (global optimum) determined by objective function evaluation, which is a set of values produced by component variables; the sounds for better evaluation can be improved through practice after practice, the values for better objective function estimation can be improved iteration by iteration.

The steps of standard harmony search are as follows:

**Step 1:** Optimization problem description and initialization of HS parameters.

The initialization of the standard HS aims to set the control parameters and to fill in the harmony memory. The main control parameters of HS are specified as follows, harmony memory size (HMS) is similar to the population size in EA; harmony memory considering rate (HMCR) determines the rate of selecting the value from the HM; pitch adjusting rate (PAR) determines the probability of local improvement; bandwidth (BW) determines the distance of adjustment; the number of improvisations (NI). The optimization problem is characterized as a function $f$ to be minimized:

$$\min f(X), X = (x_1, x_2, \dots, x_n), x_i \in [LB_i, UB_i], \quad i = 1, 2 \dots n \tag{1}$$

where $f(X)$ is the objective function. $X$ is a solution vector composed of decision variable $x_i$. $LB_i$ and $UB_i$ are lower and upper bounds for the decision variable $x_i$, respectively.

**Step 2:** Initialization of harmony memory

In this step, the HM matrix, shown in Eq. (2), is filled with randomly generated solution vectors between lower and upper bounds and ranked by the values of the objective function.

$$\begin{aligned} HM &= \begin{bmatrix} \mathrm{X}^1 & f(\mathrm{X}^1) \\ \mathrm{X}^2 & f(\mathrm{X}^2) \\ \vdots & \vdots \\ \mathrm{X}^{\mathrm{HMS}} & f(\mathrm{X}^{\mathrm{HMS}}) \end{bmatrix} \\ &= \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_n^1 & f(\mathrm{X}^1) \\ x_1^2 & x_2^2 & \cdots & x_n^2 & f(\mathrm{X}^2) \\ \vdots & \vdots & & \vdots & \vdots \\ x_1^{\mathrm{HMS}} & x_2^{\mathrm{HMS}} & \dots & x_{\mathrm{n}}^{\mathrm{HMS}} & f(\mathrm{X}^{\mathrm{HMS}}) \end{bmatrix}. \end{aligned} \tag{2}$$

**Step 3:** Improvise a new harmony

In this step, a new harmony vector $\mathrm{X}^{new} = (x_1^{new}, x_2^{new}, x_i^{new} \cdots x_n^{new})$ is generated based on three operators: (1) memory consideration; (2) pitch adjustment; (3) random selection. For a new harmony vector $\mathrm{X}^{new} = (x_1^{new}, x_2^{new}, x_i^{new} \cdots x_n^{new})$. If a component $x_i^{new}$ is randomly selected from the historical values stored in HM, it is performed as Eq. (3)

$$x_i^{new} = \begin{cases} x_i \in \{x_i^1\ x_i^2\ \cdots\ x_i^{HMS}\} & \text{if } rand < \mathrm{HMCR} \\ x_i \in X_i & \text{otherwise.} \end{cases} \tag{3}$$

Where $X_i$ $(i = 1, 2, \dots n)$ is the $i$th search space. The HMCR is the probability of assigning one value based on historical values stored in the HM, and (1 - HMCR) is the probability of randomly assigning one value according to their possible range. A HMCR value of 1.0 is not recommended because it eliminates the possibility that the solution may be improved by the values which are not stored in the HM. If the component $x_i^{new}$ comes from HM, it will be tuned with a possibility of PAR which is performed as Eq. (4)

$$x_i^{new} = \begin{cases} x_i^{new} \pm rand \times \mathrm{BW} & \text{if } rand < \mathrm{PAR} \\ x_i^{new} & \text{otherwise.} \end{cases} \tag{4}$$

Where rand is uniformly distributed number between 0 and 1 and BW is an arbitrary distance bandwidth for the continuous decision variables.

**Step 4:** Update the HM

In this step, if the newly generated harmony vector outperforms the worst one in the HM in terms of the objective function value, then the new harmony will substitute the existing worst one. The values in HM are ranked by objective function values again.

**Step 5:** Check the termination criterion

In this step, check whether the termination criterion is satisfied, if not, return Step 3 and 4 until the maximum number of iterations is reached.

**Table 1**
Solution representation of individual $X_i^t$ (LOV).

| Job, dimension | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Position, $x_{ij}^t$ | 0.3 | 0.06 | 0.9 | −0.24 | 0.85 | −0.53 |
| Sequence, $\sigma_{ij}^t$ | 3 | 4 | 1 | 5 | 2 | 6 |
| Job, $\pi_{ij}^t$ | 3 | 5 | 1 | 2 | 4 | 6 |

### 3.2. Solution representation

Because the standard HS was initially designed for solving the continuous optimization problems, therefore, the standard encoding scheme of HS cannot be directly employed for PFSSP. In order to make the standard HS suitable for solving the PFSSP, the key issue is to find an appropriate mapping rule between the job sequence and continuous harmony vector in HS. The job-permutation-based encoding scheme for PFSSP has been widely adopted in the papers. To convert individual $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$ into a job sequence $\pi_i = [\pi_{i,1}, \pi_{i,1}, \ldots, \pi_{i,n}]$, a largest-order-value (LOV) rule based on random key representation was proposed by Qian et al. (2008). Xie et al. (2014) presented a random key based largest-ranked-value (LRV). By using the LRV rule, the vector of the individual can be successfully converted to a discrete job permutation. To enable the continuous particle swarm optimization algorithm to be all classes of sequencing problems, a heuristic rule, the smallest position value (SPV) rule is developed by Tasgetiren et al. (2004). The LOV, LRV and SPV rules are widely utilized in many literatures (Li and Yin, 2013a; Liu et al., 2007). And the job sequence mapping scheme plays an important part in the evolutionary process of the algorithm, and if the mapping rule is not suitable for the PFSSP, it will increase computational time. In this paper, inspired by LOV, an efficient job sequence mapping scheme, named smallest order value (SOV), is proposed. Then, the effectiveness and efficiency of four job sequence mapping schemes are tested through experiments. Four simple examples are listed in Tables 1–3 and 19 to illustrate the different job sequence mapping schemes, respectively.

According to LOV rule, the temp sequence $\sigma_i = [\sigma_{i,1}, \sigma_{i,1}, \ldots, \sigma_{i,n}]$ is generated by ranking the individual of $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$ with descending order. Then, a discrete job permutation $\pi_i = [\pi_{i,1}, \pi_{i,1}, \ldots, \pi_{i,n}]$ is obtained by using Eq. (5)

$$\pi_{i,\sigma_{i,j}} = j. \tag{5}$$

Where $j \in [1, \ldots, n]$, and the LOV rule is illustrated with a simple example ($n = 6$) in Table 1, where the individual vector is $X_i = [0.3, 0.06, 0.9, -0.24, 0.85, -0.53]$. For the $X_i$, we can find that $x_{i,3}$ is the largest value of them, so $x_{i,3}$ is firstly selected as the first order of a job permutation. After that, $x_{i,5}$ is selected as the second one. In the similar way, $x_{i,1}, x_{i,2}, x_{i,4}$ and $x_{i,6}$ are assigned the rank values 3,4,5 and 6, respectively. Therefore, the temp sequence is $\sigma_i = [3, 4, 1, 5, 2, 6]$. According to Eq. (5), if $j = 1$, then $\sigma_{i,1} = 3$, and $\pi_{i,\sigma_{i,1}} = \pi_{i,3} = 1$; if $j = 2$, then $\sigma_{i,2} = 4$, and $\pi_{i,\sigma_{i,2}} = \pi_{i,4} = 2$; if $j = 3$, then $\sigma_{i,3} = 1$, and $\pi_{i,\sigma_{i,3}} = \pi_{i,1} = 3$; if $j = 4$, then $\sigma_{i,4} = 5$, and $\pi_{i,\sigma_{i,4}} = \pi_{i,5} = 4$; if $j = 5$, then $\sigma_{i,5} = 2$, and $\pi_{i,\sigma_{i,5}} = \pi_{i,2} = 5$; if $j = 6$, then $\sigma_{i,6} = 6$, and $\pi_{i,\sigma_{i,6}} = \pi_{i,6} = 6$. Therefore, the job sequence $\pi_i = [3, 5, 1, 2, 4, 6]$ is acquired. It is clear that LOV rule offers a simple conversion process, which makes HS suitable for solving PFSSP.

The main difference between LOV and SOV rule is that the temp sequence $\sigma_i = [\sigma_{i,1}, \sigma_{i,1}, \ldots, \sigma_{i,n}]$ is produced by ascending the individual of $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$ in SOV rule. Then, we can obtain a discrete job permutation $\pi_i = [\pi_{i,1}, \pi_{i,1}, \ldots, \pi_{i,n}]$ by using Eq. (5). A simple example is taken to show the conversion process of the SOV rule in Table 2.

According to LRV rule, all the elements of $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$ are firstly ranked by descending order so as to acquire the job permutation $\pi_i = [\pi_{i,1}, \pi_{i,1}, \ldots, \pi_{i,n}]$. In the LRV rule, the largest value of the individual vector is firstly selected and assigned the first order of the job permutation. Then, the second largest value is picked and assigned the second order. With the same way, all the values of individual vector can

**Table 2**
Solution representation of individual $X_i^t$ (SOV).

| Job, dimension | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Position, $x_{ij}^t$ | 0.3 | 0.06 | 0.9 | −0.24 | 0.85 | −0.53 |
| Sequence, $\sigma_{ij}^t$ | 4 | 3 | 6 | 2 | 5 | 1 |
| Job, $\pi_{ij}^t$ | 6 | 4 | 2 | 1 | 5 | 3 |

**Table 3**
Solution representation of individual $X_i^t$ (LRV).

| Job, dimension | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Position, $x_{ij}^t$ | 0.3 | 0.06 | 0.9 | −0.24 | 0.85 | −0.53 |
| Job, $\pi_{ij}^t$ | 3 | 4 | 1 | 5 | 2 | 6 |

**Table 4**
Solution representation of individual $X_i^t$ (SPV).

| Job, dimension | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Position, $x_{ij}^t$ | 0.3 | 0.06 | 0.9 | −0.24 | 0.85 | −0.53 |
| Job, $\pi_{ij}^t$ | 4 | 3 | 6 | 2 | 5 | 1 |

be handled to convert the vector to a job permutation. To illustrate the LRV rule, a simple instance ($n = 6$) is provided in Table 3. In the instance, because the largest value is 0.9, the dimension $j = 3$ is picked first and assigned the first order of the job permutation; then, the largest second value is 0.85, the dimension $j = 5$ is selected and assigned the second order of the job permutation. In a similar way, the job permutation $\pi_i = [3, 4, 1, 5, 2, 6]$ based on the LRV rule is obtained. As we can see, such a conversion process is really simple, and it makes HS applicable to solve the PFSSP.

For the SPV rule, it is different from the other two job sequence mapping scheme. In order to obtain a job permutation $\pi_i = [\pi_{i,1}, \pi_{i,1}, \ldots, \pi_{i,n}]$, all the elements of $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$ are firstly ranked by ascending order, which is counter to the LRV rule. In the SPV, the smallest value of the vector of individual is firstly selected and assigned the first order of the job permutation. Then, the second smallest value is selected and assigned the second order of the job permutation. In this way, all the values of the vector of individual can be handled to convert the vector to a discrete job permutation. We use a simple instance to illustrate the SPV rule in Table 4. In the instance, because the smallest value is −0.53, the dimension $j = 6$ is first picked and assigned the first order of job permutation; then, the second smallest value is −0.24, and the dimension $j = 4$ is picked and assigned the second order of the job permutation. In a similar way, all the value can be addressed to convert the vector of individual to a discrete job permutation. Based on the SPV rule, the job permutation $\pi_i = [4, 3, 6, 2, 5, 1]$ is easily acquired.

In this experiment, Carlier's benchmark set (Carlier, 1978) and Reeves and Yamada's benchmark set (Reeves and Yamada, 1998) are used to test the efficiency of the four job sequence mapping scheme. The parameters of standard HS from literature (Mahdavi et al., 2007) are set as follows: HMS = 5, HMCR = 0.9, PAR = 0.3, BW = 0.01, NI (number of iterations) = 50000. In the HS, each instance is independently executed 30 times as statistic for the solution quality. The results are shown in Table 5. In Table 5, different abbreviations are as follows, $C^*$ is the best makespan known so far, ARE denotes the average relative error to $C^*$, BRE represents the best relative error to $C^*$, $t_{avg}$ is the average time arriving at the best value or reaching the maximum number of iterations in seconds. The performance measure indicators ARE and BRE are calculated Eqs. (6) and (7).

$$ARE = \frac{A_{best} - C*}{C*} \times 100\% \tag{6}$$

$$BRE = \sum_{i=1}^{R} \left( \frac{A_i - C*}{C*} \right) \times \frac{1}{n} \times 100\% \tag{7}$$

$$WRE = \frac{A_{worst} - C*}{C*} \times 100\%. \tag{8}$$

Where $A_{best}$, $A_i$, and $A_{worst}$ are the makespan values obtained by the tested algorithm. Obviously, the smaller the percentage relative error to $C^*$, the better solution the tested algorithm generates.

**Table 5**
The results tested on different job sequence mapping schemes.

| Problem | n,m | $C^*$ | LOV | | | SOV | | | LRV | | | SPV | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | BRE | ARE | $t_{avg}$ (s) | BRE | ARE | $t_{avg}$ (s) | BRE | ARE | $t_{avg}$ (s) | BRE | ARE | $t_{avg}$ (s) |
| Car1 | 11,5 | 7038 | 0.000 | 6.301 | 0.016 | 0.000 | 2.601 | 0.072 | 0.018 | 3.282 | 0.591 | 0.036 | 3.731 | 0.577 |
| Car2 | 13,4 | 7166 | 0.391 | 4.710 | 0.743 | 0.586 | 3.713 | 0.766 | 0.684 | 4.381 | 1.129 | 1.341 | 4.579 | 1.478 |
| Car3 | 12,5 | 7312 | 0.562 | 1.457 | 1.669 | 0.352 | 2.160 | 1.066 | 1.683 | 2.003 | 2.644 | 1.350 | 1.977 | 2.469 |
| Car4 | 14,4 | 8003 | 0.000 | 3.263 | 0.051 | 0.000 | 3.463 | 0.046 | 0.227 | 3.320 | 0.832 | 0.002 | 3.151 | 0.639 |
| Car5 | 10,6 | 7720 | 0.727 | 1.215 | 1.863 | 0.508 | 1.542 | 1.473 | 0.636 | 0.853 | 2.626 | 0.713 | 0.959 | 2.422 |
| Car6 | 8,9 | 8505 | 0.584 | 2.710 | 1.471 | 0.706 | 2.140 | 1.821 | 0.623 | 2.020 | 1.985 | 0.691 | 1.969 | 2.133 |
| Car7 | 7,7 | 6590 | 0.295 | 2.335 | 1.166 | 0.107 | 3.297 | 0.734 | 0.675 | 2.505 | 1.878 | 0.578 | 2.031 | 1.790 |
| Car8 | 8,8 | 8366 | 0.000 | 2.542 | 0.061 | 0.000 | 2.364 | 0.087 | 0.582 | 1.529 | 2.082 | 0.375 | 1.460 | 1.627 |
| Rec01 | 20,5 | 1247 | 0.732 | 1.354 | 2.972 | 0.580 | 1.190 | 2.533 | 6.017 | 6.679 | 3.146 | 5.130 | 6.113 | 2.897 |
| Rec03 | 20,5 | 1109 | 0.682 | 1.286 | 2.796 | 0.511 | 1.118 | 2.682 | 2.146 | 2.872 | 3.199 | 1.969 | 2.820 | 2.977 |
| Rec05 | 20,5 | 1242 | 0.569 | 0.859 | 2.784 | 0.534 | 0.739 | 2.626 | 1.812 | 2.283 | 3.307 | 2.112 | 2.443 | 2.874 |
| Rec07 | 20,10 | 1566 | 1.362 | 1.648 | 3.813 | 1.328 | 1.715 | 3.131 | 3.491 | 4.304 | 3.753 | 3.508 | 4.291 | 3.290 |
| Rec09 | 20,10 | 1537 | 1.980 | 2.556 | 3.661 | 1.917 | 2.380 | 3.099 | 4.218 | 5.155 | 3.507 | 3.917 | 4.800 | 3.334 |
| Rec11 | 20,10 | 1431 | 1.694 | 2.687 | 3.064 | 2.213 | 2.581 | 3.245 | 6.112 | 7.211 | 3.482 | 5.516 | 6.513 | 3.317 |
| Rec13 | 20,15 | 1930 | 1.883 | 2.271 | 3.766 | 2.117 | 2.610 | 3.524 | 5.152 | 6.004 | 4.080 | 4.603 | 5.443 | 3.845 |
| Rec15 | 20,15 | 1950 | 2.462 | 2.761 | 3.867 | 2.109 | 2.596 | 3.866 | 4.233 | 4.913 | 3.996 | 3.615 | 4.489 | 3.715 |
| Rec17 | 20,15 | 1902 | 3.062 | 3.580 | 3.515 | 3.055 | 3.437 | 4.005 | 5.666 | 6.432 | 3.853 | 5.957 | 6.844 | 3.469 |
| Rec19 | 20,30 | 2093 | 2.599 | 3.360 | 3.663 | 3.131 | 3.781 | 3.645 | 6.934 | 8.136 | 3.991 | 6.800 | 8.218 | 3.524 |
| Rec21 | 20,30 | 2017 | 2.264 | 2.912 | 3.697 | 2.352 | 3.055 | 3.360 | 7.108 | 8.008 | 3.799 | 6.753 | 7.796 | 3.561 |
| Rec23 | 20,30 | 2011 | 3.063 | 3.647 | 3.553 | 2.743 | 3.377 | 3.546 | 7.076 | 8.266 | 3.885 | 7.005 | 8.292 | 3.575 |
| Rec25 | 30,15 | 2513 | 3.774 | 4.448 | 4.110 | 3.548 | 4.148 | 3.995 | 7.335 | 8.410 | 4.288 | 7.500 | 8.596 | 4.034 |
| Rec27 | 30,15 | 2373 | 3.110 | 3.753 | 4.141 | 2.963 | 3.640 | 4.118 | 6.747 | 7.783 | 4.180 | 6.507 | 7.670 | 4.135 |
| Rec29 | 30,15 | 2287 | 4.117 | 5.062 | 4.061 | 4.148 | 5.065 | 3.796 | 9.541 | 11.055 | 4.372 | 9.560 | 10.952 | 4.119 |
| Rec31 | 50,10 | 3045 | 4.785 | 5.717 | 4.277 | 4.645 | 5.629 | 4.154 | 11.237 | 12.277 | 4.707 | 11.126 | 12.113 | 4.577 |
| Rec33 | 50,10 | 3114 | 1.788 | 2.548 | 4.474 | 1.845 | 2.571 | 4.088 | 8.047 | 9.138 | 4.620 | 7.683 | 8.922 | 4.385 |
| Rec35 | 50,10 | 3277 | 0.450 | 1.219 | 4.032 | 0.339 | 0.906 | 3.740 | 4.184 | 5.169 | 4.751 | 4.620 | 5.394 | 4.527 |
| Rec37 | 75,20 | 4951 | 8.010 | 9.077 | 7.211 | 7.966 | 9.081 | 6.871 | 16.051 | 16.654 | 7.633 | 16.278 | 16.853 | 7.400 |
| Rec39 | 75,20 | 5087 | 10.302 | 11.318 | 10.559 | 10.268 | 11.258 | 10.665 | 17.656 | 18.339 | 12.542 | 17.552 | 18.311 | 11.347 |
| Rec41 | 75,20 | 4960 | 12.126 | 13.190 | 11.279 | 12.183 | 13.173 | 10.394 | 20.455 | 21.162 | 11.113 | 20.182 | 21.043 | 11.101 |
| Average | | | 2.530 | 3.786 | 3.529 | 2.509 | 3.632 | 3.350 | 5.736 | 6.901 | 3.999 | 5.620 | 6.820 | 3.763 |

**Table 6**
The ranking results with different indicators.

| Rules | LOV | SOV | LRV | SPV |
|---|---|---|---|---|
| BRE | 2.530 | 2.509 | 5.736 | 5.620 |
| Rank (BRE) | 2 | 1 | 4 | 3 |
| ARE | 3.786 | 3.632 | 6.901 | 6.820 |
| Rank (ARE) | 2 | 1 | 4 | 3 |
| $t_{avg}$ (s) | 3.529 | 3.350 | 3.999 | 3.763 |
| Rank ($t_{avg}$) | 2 | 1 | 4 | 3 |

From Table 6, it can be seen that the SOV rule is ranked first no matter what kind of performance indicators to rank. This indicates the effectiveness and efficiency of the SOV rules. In fact, the four job sequence mapping schemes can be divided into two groups, LOV and SOV rules are the group one. The others are the group two. The values in the group one are far better than those in the group two. However, the solutions obtained through SOV rule are a bit better than those of the LOV rule.

### 3.3. Population initialization

#### 3.3.1. NEH heuristic

The NEH heuristic developed by Nawaz et al. (1983), has been proved to be one of the effective heuristics for the PFFSP (Liu et al., 2007, 2014). The basic idea of the NEH heuristic is that jobs with higher total processing time on all machines should be processed than those with less total processing time. In general, the HEH heuristic has two important features: (1) all the jobs are ranked in non-increasing according to their total processing time on all machines. (2) Based on the first feature, a job permutation is obtained by assessing the partial sequences. The steps of the NEH heuristic are depicted as follows.

**Step 1:** According to their total processing time on all machines, all the jobs are sorted with non-increasing order. Then, a job sequence $\pi(j) = [\pi(1), \pi(2), \ldots, \pi(n)]$ is obtained.

**Table 7**
Conversion of the job permutation to the individual vector.

| Job, dimension $j$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Permutation by NEH | 2 | 6 | 3 | 5 | 1 | 4 |
| $sq_{NEH,j}$ | 5 | 1 | 3 | 6 | 4 | 2 |
| Individual value | 0.3 | −0.9 | −0.3 | 0.6 | 0 | −0.6 |

**Step 2:** The first two jobs $\pi(1)$ and $\pi(2)$ are taken, and the possible schedules of these two jobs are evaluated. Then, the optimal sequence is selected as the current sequence.

**Step 3:** Take job $\pi(j)$, $j = 3, 4, \ldots, n$, and find the optimal sequence by placing it in all possible positions of the partial sequence of jobs which have been scheduled. Then the optimal sequence will be chosen for the next generation. For the partial sequences which have the same fitness value, any sequence can be chosen as the optimal one. Repeat this step until all the jobs are scheduled.

Because the solution generated by NEH heuristic is a discrete job permutation, in order to make it perform HS search, the job permutation is supposed to convert into position values of a certain individual. By using the following Eq. (9), the conversion process is executed:

$$x_{NEH,j} = x_{min,j} + \frac{\left(x_{max,j} - x_{min,j}\right)}{n} \times \left(sq_{NEH,j} - 1\right), \quad j = 1, 2, \ldots n. \tag{9}$$

Where $sq_{NEH,j}$ is the job index in the $j$th dimension of the job permutation. $x_{max,j}$ and $x_{min,j}$ are the upper and lower bounds of the individual value. $x_{NEH,j}$ is the individual value in the $j$th dimension of the individual vector. Suppose the job permutation obtained by NEH heuristic is $\pi = [2, 6, 3, 5, 1, 4]$. Because the index of first processed job is 5, and the index of the second processed job is 1. In a similar way, the index sequence is $sq_{NEH,j} = [5, 1, 3, 6, 4, 2]$. Then, according to Eq. (9), the individual vector will be $X = [0.3, -0.9, -0.3, 0.6, 0, -0.6]$. A simple instance ($n = 6$) is offered to illustrate the conversion process in Table 7.

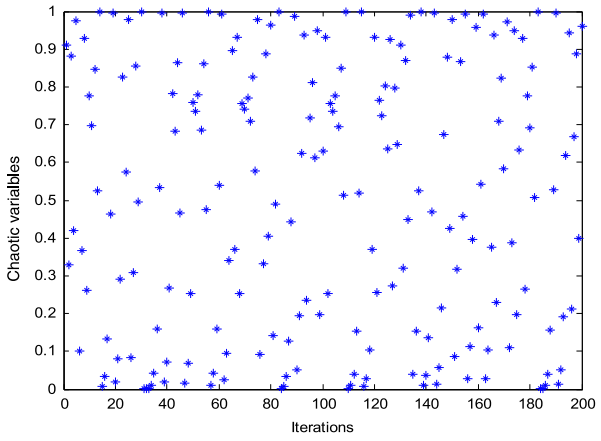


**Fig. 1.** Logistic map with a random initial value.

### 3.3.2. Chaotic sequence using logistic map

Chaos, which is non-period, non-converging and bounded, has been applied in variety of research fields since the first introduction of canonical chaotic attractor by Lorenz (1963). Chao is a deterministic and pseudo-randomness process appeared in non-linear dynamic system. And Chaos is a bounded unstable dynamic behavior that can provide search diversity in an optimization procedure. Chaos variables have good ergodic properties with the increase of iteration times. This kind of characteristic is very useful for global search. One of important features of the chaos is that it is very sensitive to the initial point and the small changes for the initial condition can result in huge different future behaviors. Lately, chaotic sequences have been employed instead of random sequences and many satisfying results have been gained in the applications (Arul et al., 2013; Yuan et al., 2014). By combining with heuristic optimization algorithms, chaotic sequences can strengthen searching behavior and avoid being trapped into local optimum. Among all the chaotic maps, the logistic map introduced by May (1976) has drawn increasing attention to chaos researchers due to its simplicity in implement and excellent dynamical behavior. Meanwhile, the logistic map can be efficiently cooperated with heuristic optimization algorithms so as to enhance the ability of searching for global optima. Moreover, the average computational times of logistic map is less than those of well-known chaotic maps (Tavazoei and Haeri, 2007). The logistic map is formally defined as follows:

$$Y_{n+1} = \varphi * Y_n * \left(1 - Y_n\right). \tag{10}$$

Where $n$ represents the maximum number of iterations and $Y_n$ is the $n$ th chaotic variable. $\varphi \in [0,4]$ is a logistic control parameter and the variable $Y_n$ chaotically runs in [0,1] on the condition that the initial value $Y_n \in (0,1)$ and that $Y_n \notin [0.0, 0.25, 0.5, 0.75, 1.0]$. The chaotic sequences will appear when $\varphi = 4$. So $\varphi = 4$ is used in the experiments. The behaviors of chaotic variables using logistic map with 200 iterations is shown in Fig. 1, where the random number is chosen as the initial value.

### 3.4. Local search based on VNS

Variable neighborhood search (Hansen and Mladenović, 2001) is a simple and effective meta-heuristic for combinatorial and global optimization problems. The basic ideal of VNS is systematic change, both in the descent to search for local optimum and in the perturbation to escape from valleys. VNS, as the local search methodology, has been widely used in many literatures (Chen et al., 2015; Li and Yin, 2013b; Zheng and Yamashiro, 2010), and the performance of hybridization with VNS is obvious. For the VNS, there are varieties of neighborhoods to choose such as inert, swap, 2-opt, interchange and inverse. Among all the neighborhoods, insert and swap are more effective and efficient than others. Therefore, the two neighborhoods insert and swap are adopted in this paper to enhance the local search ability. The operations of the two neighborhoods are showed in Fig. 2. From Fig. 2, it is clear that the insert structure includes two operations. Randomly choose two different positions $r_1$ and $r_2$ in a job permutation, if $r_1 < r_2$, $r_1$ will be inserted the back of $r_2$. Otherwise, $r_1$ will be inserted in front of $r_2$. The local search based on VNS is described in detail as shown in Algorithm 1.

**Algorithm 1.** Local search by using variable neighborhood search

```
According to the SOV rule, convert the individual X_i to a discrete job sequence π_i.
For i=1 to n× (n-1) do
  Randomly choose u and v, where u ≠ v
  π_new = insert(π_i, u, v)
    If f(π_new) < f(π_i) then
      π_i = π_new
    end if
  end for
Return π_i and π_i will be used as the initial sequence in the swap operation.
For j=1 to n x (n-1) do
    Randomly choose u and v, where u ≠ v
    π_new = swap(π_i, u, v)
    If f(π_new) < f(π_i) then
        π_i = π_new
    end if
end for
Return π_i
```

### 3.5. The procedure of HHS

According to the above of job sequence mapping scheme, NEH, HS, and VNS, a hybrid harmony search (HHS) algorithm is presented to deal with PFSSP. The steps of HHS are described in detail as follows and the procedure is shown in Algorithm 3.

**Step 1:** Initialization of HHS parameters

In the proposed algorithm, four parameters are needed to initialize. HMS = 30, HMCR = 0.9, PAR = 0.2, NI = 500.

**Step 2**: Harmony memory initialization

In this step, initialization is yielded by two-phase procedure: (1) an initialization procedure based on famous NEH heuristic algorithm is applied to generate one solution; (2) the rest are initialized by chaotic sequence produced by logistic map instead of random population initialization. Chaotic sequence can guarantee the diversity of initialization of harmony memory, accelerate its convergence performance and reinforce the global search ability. The chaotic initialization is depicted as Algorithm 2.

**Algorithm 2.** Initialization with chaos by using logistic map

```
For i=1 to HMS do
  Randomly select the initial chaotic variable;
  For j=1 to N do
    Generate chaotic variables cv_{i,j} by using logistic map;
    x_{i,j} = x_j^min + cv_{i,j} * (x_j^max − x_j^min)
  end for
end for
```

**Step 3:** Improvise a new harmony

In this step, the new harmony $X^{new}$ of HHS still comes from three phases as the standard HS algorithm does. However, the difference between the two algorithms is the pitch adjustment. In the HHS, the current best harmony vector is adopted to speed up the convergence performance and help to find the promising solutions. The other two

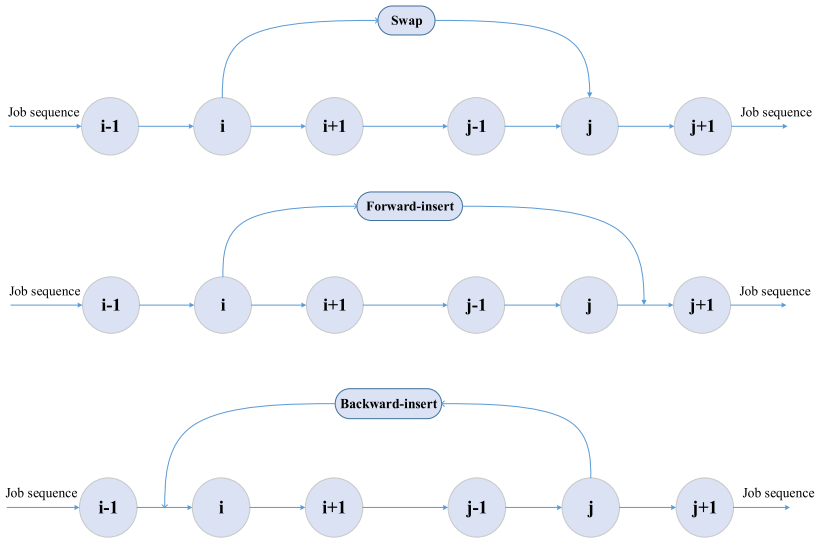


**Fig. 2.** The neighborhood structures for VNS.

phases are the same as the standard HS. The pitch adjustment is performed as Eq. (11).

$$x_i^{new} = x_i^{new} + rand \times \left(x_i^{best} - x_i^{new}\right) \quad i = 1, 2, \ldots, n. \tag{11}$$

**Step 4:** Update the harmony memory

In this step, if the newly generated harmony vector $x^{new}$ outperforms the worst one in the HM in terms of makespan, a local search is performed and returns $x_L^{new}$, then $x_L^{new}$ will substitute the existing worst one. Otherwise, Let the $x^{new}$ firstly learns from the best harmony $x^{best}$ in the current HM and returns $x^{new'}$, and then execute local search for $x^{new'}$ and return $x_L^{new'}$. The existing worst one is replaced by $x_L^{new'}$.

**Step 5:** Check the termination criterion

In this step, check whether the termination criterion is satisfied, if not, return Step 3 and 4 until the maximum number of iterations is reached.

For the above procedure, two stop criterions are adopted: (1) the number of generations equals the maximum number of generations; (2) the makespan reaches the optimum or the lower bound known so far.

## 4. Computational results and comparisons

### *4.1. Computational results and comparisons in the continuous optimization problems*

In this part, to verify the performance of HHS in the continuous optimization, ten typical benchmark test functions, which come from Liang et al. (2013) and Suganthan et al. (2005), are employed and listed in Table 20 in "Appendix B". These benchmarks are either unimodal or multimodal. The associated solutions in terms of the best, worst, mean, standard deviation (Std) and CPU running time ($t_{avg}$) are reported in Table 8. The unimodal benchmarks are chosen to verify the convergence speed and accuracy, while multimodal functions are selected to testify the global searching ability. For the multimodal function, it has many local optima and the number of local optima exponentially increases with dimension size, therefore, it is easy to trap into local optima.

In this experimental study, all the functions are tested in fifty dimensions. The parameters of HHS algorithms are set as follows: HMS = 5, HMCR = 0.9950 and PAR = 0.4. In order to ensure the reliability of experimental results, HHS and all the compared algorithms are run thirty times independently for each case. This version of MATLAB considers the numbers smaller than 4.9407e−324 as zero. The best solutions (smallest is best) are highlighted in bold.

To comprehensively verify the effectiveness of HHS algorithm, this paper selects classical variants of HS, state-of-the-art HS variants and other EAs (Eberhart and Kennedy, 1995; Mahdavi et al., 2007; Omran and Mahdavi, 2008; Storn and Price, 1997; Valian et al., 2014; Zou et al., 2010a, b). All the compared algorithms in this part are implemented by ourselves. Besides, some cases of convergence graphs of HS, GHS, IGHS, PSO, DE and HHS are shown in Fig. 3 so as to demonstrate the convergence speed of HHS more clearly. From Table 8, it can be observed that the results obtained by HHS are better than the other compared algorithms on almost all the benchmarks except for $f4$ and $f7$. To our surprise, HHS can find global optimal values on test functions $f1, f2, f6, f9$ with D = 50, which also have fast convergence speed as shown in Fig. 3. For the $f4$, the results acquired by IGHS and HHS are the same order, which implicitly suggests there is little obvious differences between them. However, from Fig. 3, it can be found that the convergence speed of HHS is faster than that of IGHS. For the $f7$, the solution result achieved by IGHS is ranking first and outperforms the other algorithms. According to the above observations and analysis, HHS exhibits the extremely convergence performance. And the experimental results demonstrate that it outperforms conspicuously than the aforementioned variants of HS and other EAs, which verifies the effectiveness and efficiency of HHS in the continuous optimizations.

### *4.2. Computational results and comparisons in the combinatorial optimization problems*

To verify the performance of the HHS algorithm for PFSSP, 149 benchmark instances with different scales from the OR-library (Beasley, 1990) are adopted. These benchmarks are divided into three categories:

(1) The first eight instances Car01–Car08 were designed by Carlier (1978).
(2) The second twenty-one instances Rec01–Rec41 were designed by Reeves and Yamada (1998).
(3) The third 120 instances from Taillard (1993).

The experiments were implemented on an Intel 2.53 GHZ Core (TM) i3 processor with 2 GB of RAM where HHS algorithm were programmed with MATLAB (2010b) under Win7 (64X). For the instances $200 \times 10$, $200 \times 20$, and $500 \times 20$ are executed 10 times and each of the rest instances is independently executed 20 times as statistic for the solution quality. The boldface in each table denotes the better solutions yielded by corresponding compared algorithms.

**Table 8**
The comparison results of HS, IHS, GHS, NGHS, PSO, DE, IGHS and HHS (D = 50).

| Functions | Algorithms | Best | Mean | Worst | Std | $t_{avg}$ (s) |
|---|---|---|---|---|---|---|
| $f1$ | HS | 2.8071e+02 | 5.0822e+02 | 7.4202e+02 | 1.2107e+02 | 1.2858 |
| | IHS | 4.2397e+02 | 5.3862e+02 | 7.6877e+02 | 1.0303e+02 | 1.5692 |
| | GHS | 8.8011e−03 | 2.1301e+00 | 1.2659e+01 | 3.0908e+00 | 1.3822 |
| | NGHS | 1.3540e−01 | 2.3866e+00 | 6.1969e+00 | 1.7960e+00 | 1.0915 |
| | IGHS | **0** | 4.0082e−310 | 8.0165e−309 | 0 | 1.3143 |
| | PSO | 3.7200e−02 | 5.1312e−02 | 6.5301e−02 | 1.9902e−02 | 2.5940 |
| | DE | 2.6012e−03 | 4.8109e−03 | 9.3091e−03 | 1.8113e−03 | 3.0448 |
| | HHS | **0** | **0** | 0 | 0 | 0.3501 |
| $f2$ | HS | 7.7532e+00 | 9.7372e+00 | 12.4736e+00 | 1.2339e+00 | 1.1909 |
| | IHS | 8.1465e+00 | 10.0049e+00 | 11.5264e+00 | 1.0132e+00 | 1.4750 |
| | GHS | 9.7349e−04 | 3.2160e−01 | 1.1752e+00 | 3.5051e+00 | 1.2804 |
| | NGHS | 1.3030e−01 | 2.7480e−01 | 5.3202e−01 | 1.1903e−01 | 1.0868 |
| | IGHS | 3.9569e−251 | 1.5897e−239 | 2.3361e−238 | 1.2897e−232 | 1.3220 |
| | PSO | 1.1453e+00 | 3.0630e+00 | 6.5980e+00 | 1.1923e+00 | 2.5688 |
| | DE | 2.6534e−02 | 4.5174e−02 | 6.5111e−02 | 1.2756e−02 | 3.1462 |
| | HHS | **0** | **0** | 0 | 0 | 0.6815 |
| $f3$ | HS | 1.6081e+04 | 2.7814e+04 | 3.7176e+04 | 6.1281e+03 | 7.4336 |
| | IHS | 1.6096e+04 | 2.8581e+04 | 5.0024e+04 | 7.7283e+03 | 7.6318 |
| | GHS | 2.8690e+04 | 6.3641e+04 | 8.3398e+04 | 1.3697e+04 | 7.5043 |
| | NGHS | 2.5284e+04 | 3.3404e+04 | 4.7183e+04 | 6.1816e+03 | 7.2999 |
| | IGHS | 1.0925e+02 | 1.3273e+04 | 4.1243e+04 | 1.2923e+04 | 7.5834 |
| | PSO | 1.0370e+01 | 2.2628e+01 | 3.6552e+01 | 7.1700e+00 | 8.8885 |
| | DE | 1.8217e+03 | 3.4163e+03 | 4.8987e+03 | 9.3387e+02 | 21.4171 |
| | HHS | **5.1963e−212** | **2.6139e−78** | 4.3476e−77 | 9.7131e−78 | 6.3629 |
| $f4$ | HS | 1.4603e+04 | 2.9832e+04 | 5.0311e+04 | 1.0072e+04 | 1.2084 |
| | IHS | 1.0148e+04 | 2.5154e+04 | 5.0482e+04 | 1.0422e+04 | 1.5012 |
| | GHS | 9.2761e+00 | 2.3298e+02 | 1.2454e+03 | 2.9875e+02 | 1.3499 |
| | NGHS | 5.5858e+02 | 1.2546e+03 | 3.6749e+03 | 7.0284e+02 | 1.0786 |
| | IGHS | **5.9769e−29** | **3.1354e+01** | 4.8457e+01 | 2.3605e+01 | 1.2763 |
| | PSO | 4.8785e+01 | 9.9229e+01 | 2.2495e+02 | 5.4466e+01 | 2.6180 |
| | DE | 4.8662e+01 | 1.2952e+02 | 2.5098e+02 | 5.2484e+01 | 3.2532 |
| | HHS | 4.8538e+01 | 4.8665e+01 | 4.8851e+01 | 8.5100e−02 | 1.4877 |
| $f5$ | HS | 3.8091e−01 | 4.1317e−01 | 4.3133e−01 | 1.3709e−02 | 1.8521 |
| | IHS | 3.9893e−01 | 4.1792e−01 | 4.3500e−01 | 1.0609e−02 | 2.1348 |
| | GHS | 1.9003e−02 | 1.1765e−01 | 2.2228e−01 | 5.3807e−02 | 1.9516 |
| | NGHS | 3.8091e−01 | 4.2436e−01 | 4.5233e−01 | 1.6400e−02 | 1.6236 |
| | IGHS | 4.9024e−03 | 2.5612e−02 | 4.0823e−02 | 7.6219e−03 | 2.0270 |
| | PSO | 2.7218e−01 | 3.0937e−01 | 3.5824e−01 | 2.0903e−03 | 3.4986 |
| | DE | 2.2277e−01 | 2.5855e−01 | 3.1280e−01 | 2.5664e−02 | 4.5908 |
| | HHS | **4.9105e−03** | **9.8239e−03** | 1.0190e−02 | 6.9091e−03 | 1.8494 |
| $f6$ | IHS | 3.2928e+01 | 4.4569e+01 | 5.5632e+01 | 5.9842e+00 | 1.7189 |
| | GHS | 3.1258e−04 | 6.1341e−01 | 5.2396e+00 | 1.5263e+00 | 1.5342 |
| | NGHS | 3.6711e−02 | 2.1409e+00 | 6.4158e+00 | 1.5158e+00 | 1.2886 |
| | IGHS | **0** | 4.5609e+01 | 1.5719e+02 | 5.9747e+01 | 0.9272 |
| | PSO | 2.0603e+01 | 5.4268e+01 | 8.8555e+01 | 2.0341e+01 | 2.8752 |
| | DE | 3.1530e+02 | 3.5556e+02 | 3.8637e+02 | 1.9788e+01 | 3.9621 |
| | HHS | **0** | **0** | 0 | 0 | 0.0310 |
| $f7$ | HS | 5.9946e+02 | 9.0901e+02 | 1.3011e+03 | 2.1964e+02 | 1.7935 |
| | IHS | 6.2151e+02 | 8.2164e+02 | 9.9135e+02 | 1.1020e+02 | 1.9932 |
| | GHS | 8.1012e−03 | 2.4834e+00 | 8.2113e+00 | 2.9199e+00 | 1.9123 |
| | NGHS | 8.2612e−02 | 6.1398e+01 | 2.5691e+01 | 6.1921 | 1.5850 |
| | IGHS | **6.3638e−04** | **6.3638e−04** | 6.3638e−04 | 0 | 1.6927 |
| | PSO | 1.5683e+04 | 1.6749e+04 | 1.7783e+04 | 5.5178e+02 | 3.3557 |
| | DE | 1.3432e+04 | 1.4524e+04 | 1.5436e+04 | 5.4187e+02 | 3.6751 |
| | HHS | 1.1425e+04 | 1.2612e+04 | 1.3367e+04 | 5.3338e+02 | 1.9897 |
| $f8$ | HS | 4.5531e+00 | 5.3964e+00 | 6.1890e+00 | 4.6801e+00 | 1.8688 |
| | IHS | 4.6680e+00 | 5.2827e+00 | 5.8560e+00 | 3.3973e−01 | 2.2145 |
| | GHS | 1.0301e−02 | 3.1521e−01 | 1.7082e+00 | 3.9971e−01 | 1.9321 |
| | NGHS | 1.9382e−01 | 9.5271e−01 | 1.8407e+00 | 3.6091e−01 | 1.6449 |
| | IGHS | **4.4409e−15** | 7.9936e−15 | 1.5099e−14 | 1.9965e−15 | 1.7832 |
| | PSO | 2.5950e+00 | 3.6058e+00 | 5.0122e+00 | 7.3551e−01 | 3.4380 |
| | DE | 9.3091e−03 | 1.4901e−02 | 1.9001e−02 | 2.8120e−03 | 4.5249 |
| | HHS | **4.4409e−15** | **4.4409e−15** | 4.4409e−15 | 0 | 1.7321 |
| $f9$ | HS | 4.3386e+00 | 6.1543e+00 | 8.4473e+00 | 1.2602e+00 | 3.4288 |
| | IHS | 4.6275e+00 | 5.9968e+00 | 8.1187e+00 | 9.6972e−01 | 3.8518 |
| | GHS | 5.8272e−05 | 6.7331e−01 | 1.1793e+00 | 4.1201e−01 | 3.4492 |
| | NGHS | 4.2861e−01 | 8.1673e−01 | 1.0452e+00 | 1.8312e−01 | 3.2390 |
| | IGHS | **0** | **0** | 0 | 0 | 0.3206 |
| | PSO | 9.5984e+01 | 1.1359e+02 | 1.3325e+02 | 8.8694e+00 | 4.9782 |
| | DE | 1.7329e−03 | 8.3214e−03 | 2.5532e−02 | 6.1109e−03 | 9.0524 |
| | HHS | **0** | **0** | 0 | 0 | 0.0785 |

(*continued on next page*)

**Table 8** (*continued*)

| Functions | Algorithms | Best | Mean | Worst | Std | $t_{avg}$ (s) |
|---|---|---|---|---|---|---|
| $f10$ | HS | 4.4398e+04 | 5.6644e+04 | 7.9366e+04 | 9.9031e+03 | 1.2672 |
| | IHS | 4.2591e+04 | 5.4447e+04 | 7.5241e+04 | 8.2673e+03 | 1.5775 |
| | GHS | 7.5240e+03 | 2.7461e+04 | 4.9664e+04 | 1.0660e+04 | 1.3906 |
| | NGHS | 8.3647e+04 | 1.2005e+05 | 1.6712e+05 | 1.9545e+04 | 1.1430 |
| | IGHS | 3.1457e+02 | 5.1580e+03 | 1.3824e+04 | 4.1422e+03 | 1.3604 |
| | PSO | 1.0155e+02 | 5.8813e+02 | 1.6840e+03 | 4.4558e+02 | 2.6693 |
| | DE | 4.9276e+04 | 6.8121e+04 | 7.9205e+04 | 8.4196e+03 | 3.2764 |
| | HHS | **9.6243e–77** | **2.8110e–14** | 5.6192e–13 | 1.2565e–14 | 1.1186 |

**Algorithm 3.** HHS for the PFSSP

Initial the parameters including HMS, HMCR, PAR and NI.
Use the NEH heuristic and logistic chaos to generate initial population $X_i^0$ and fill in HM.
Apply the SOV rule to convert the harmony individual $X_i^0$ into a discrete job sequence
$\pi_i^0 = [\pi_{i,1}^0, \pi_{i,2}^0, \cdots, \pi_{i,n}^0]$, where $n$ is the number of jobs in the PFSSP. $i \in [1, \mathrm{HMS}]$
Evaluate the makespan of the harmony individuals in harmony memory.
**For** $t$=1 to NI **do**
  **For** $i$=1 to $n$ **do**
    **If** $r_1 < \mathrm{HMCR}$ **then**
      **If** $r_2 < 0.5$ **then**
        $x_i^{new} = x_{\max}^i + x_{\min}^i - x_i^j \quad where\ j = 1, 2, \cdots, \mathrm{HMS}$ // opposition-based learning
      **else**
        $x_i^{new} = x_i^j$
      **end if**
      **If** $r_3 < \mathrm{PAR}$ **then**
        $x_i^{new} = x_i^{new} + r_4 \times (x_i^{best} - x_i^{new})$
      **end if**
    **else**
      $x_i^{new} = x_{\min}^i + r_5 \times (x_{\max}^i - x_{\min}^i)$
    **end if**
  **end for**
  Apply the SOV rule convert the new harmony vector $x_t^{new}$ to a job permutation $\pi_t^{new}$.
  Apply the SOV rule convert the worst harmony vector $x_t^{worst}$ to a job permutation $\pi_t^{worst}$.
  **If** $f(\pi_t^{new}) < f(\pi_t^{worst})$ **do**
    Perform local search for the newly generated harmony individual $\pi_t^{new}$. Return $\pi_{L,t}^{new}$
    $\pi_t^{worst} = \pi_{L,t}^{new}$ where $\pi_{L,t}^{new}$ is new permutation after performing the local search.
    Convert the job permutation to the harmony individual and store in the HM. // See Eq.(9)
  **else**
    $x_t^{new'} = x_t^{new} + r_6 \times (x_t^{best} - x_t^{new})$
    Apply the SOV rule convert the new harmony vector $x_t^{new}$ to a job permutation $\pi_t^{new'}$.
    Perform local search for the newly generated harmony individual $\pi_t^{new}$. Return $\pi_{L,t}^{new'}$
    $\pi_t^{worst} = \pi_{L,t}^{new'}$
    Convert the job permutation to the harmony individual and store in the HM. // See Eq.(9)
  **end if**
  Update harmony memory again and evaluate the fitness value for the *(t+1)th* generation.
  $r_1 - r_6$ are uniformly distributed numbers between 0 and 1.
  $f$ is the function of calculating makespan.
**end for**

#### 4.2.1. Parameters analysis

The set of parameters greatly affects the performance of HHS. To test the influence of the parameters in HHS, the famous Taguchi method (Montgomery, 2008) of design for the experiments is utilized to analyze the parametric sensitivity. In the HHS, there are three important parameters: harmony memory size (HMS), harmony memory considering rate (HMCR), pitch adjusting rate (PAR). In this experiment, the instance Rec19, which is very difficult to achieve optimal solution, is chosen as the benchmark case. Table 9 lists the corresponding values of different parameters in HHS. In Table 10, the parameter combinations

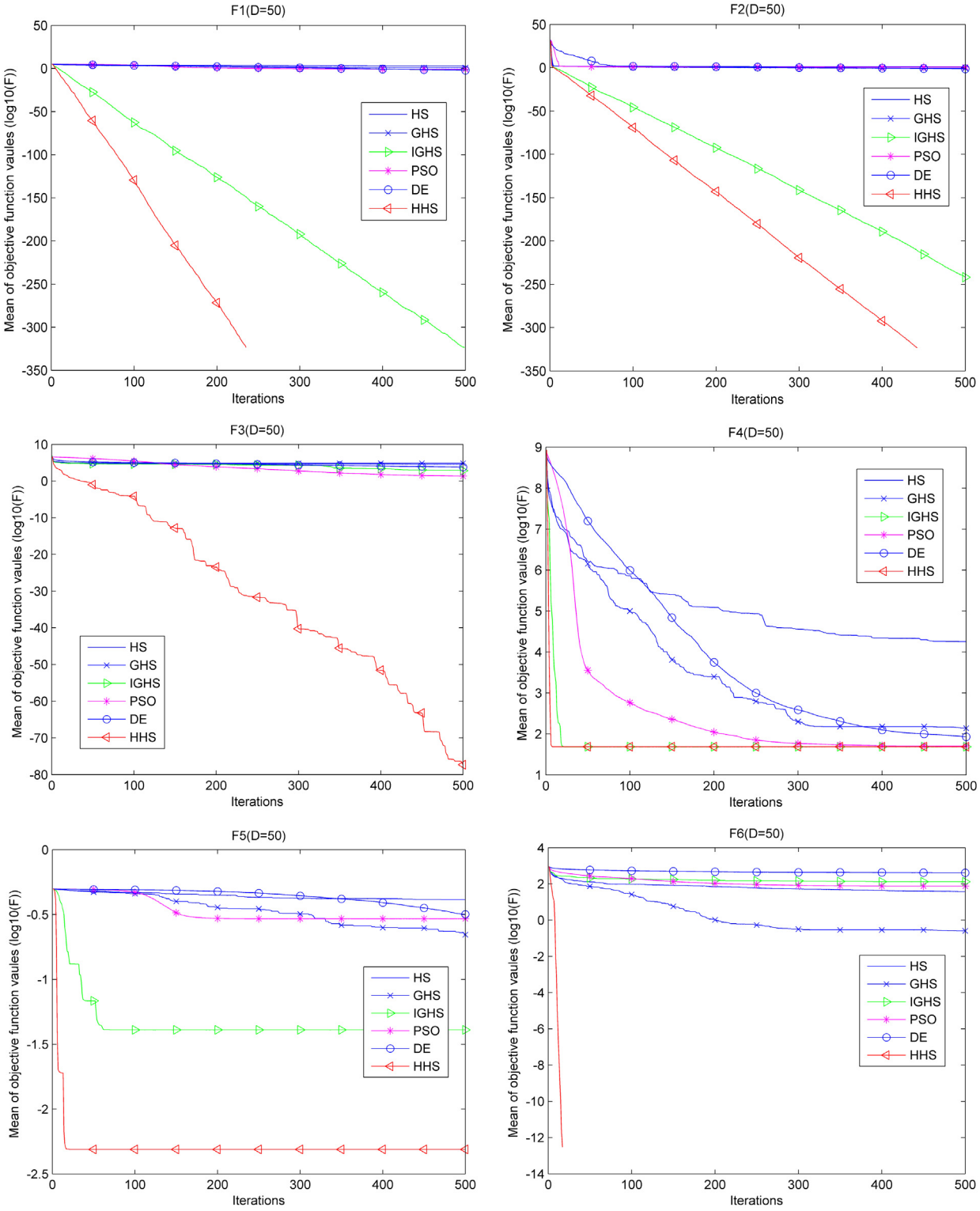


**Fig. 3.** Convergence graphs of test functions.

and ARE (See Eq. (6)) obtained by HHS are listed. According to Table 10, the significance of each parameter is described in Table 11. Furthermore, the change trend of each parameter for different value is shown in Fig. 4. Seen from Fig. 4, it is clear that PAR has a big change trend, which means that PAR is more crucial to HHS than the other two parameters. A small PAR is good for improving the solution accuracy and speeding up convergence performance. A large PAR could lead the algorithm to search in the global domain. Besides, HMCR ranks the second place, which implies it is also an important factor for HHS. A large HMCR can diversify the population, especially in the late evolutionary process. A small HMCR can lead to premature convergence. However, for the HMS, the change trend is stable, which means it has slight effect on the performance of HHS. A small HMS could slow the convergence speed and lead to poor diversity in the HM. A large HMS will cause an oversize computational budget. According to the above

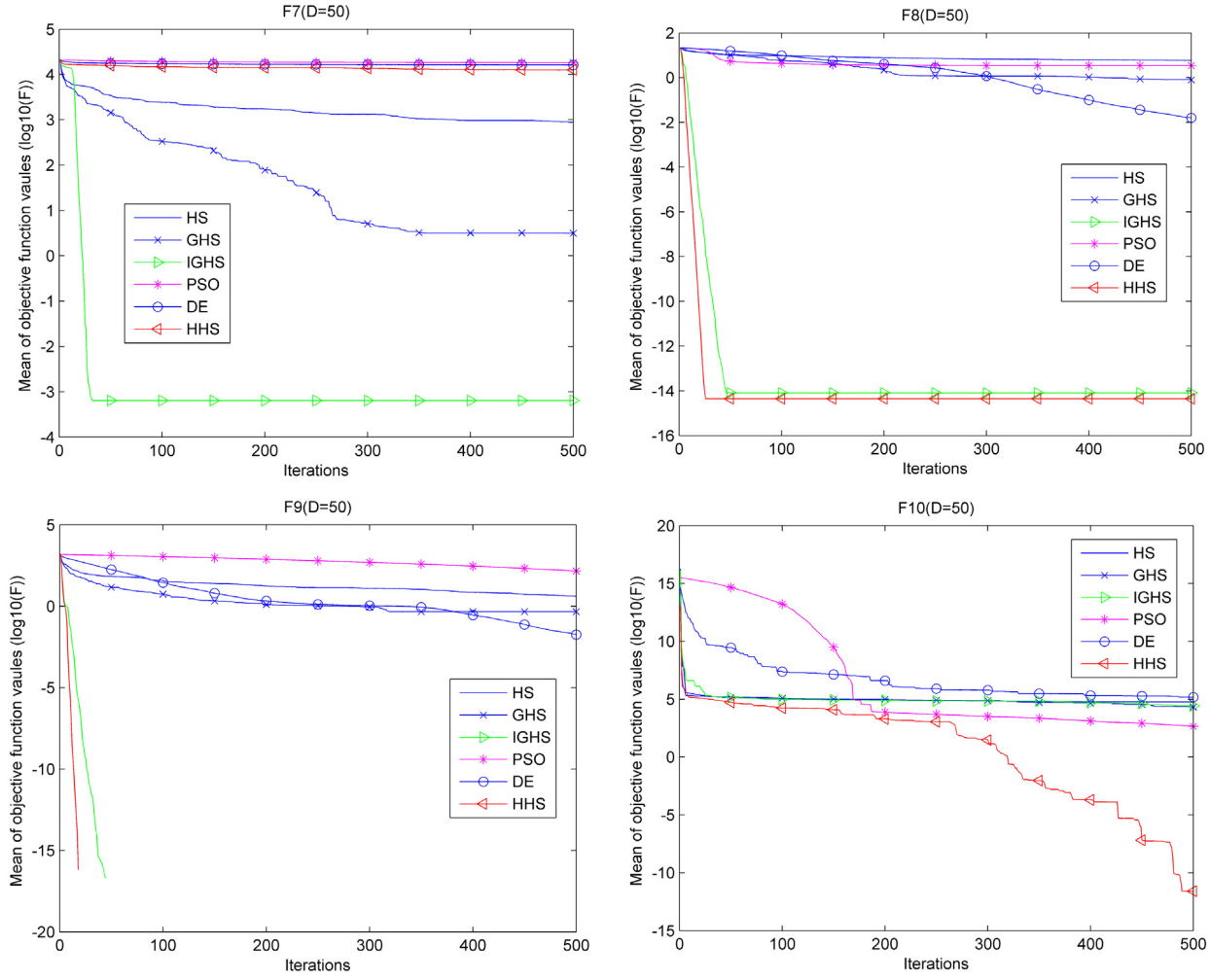


**Fig. 3.** (*continued*)

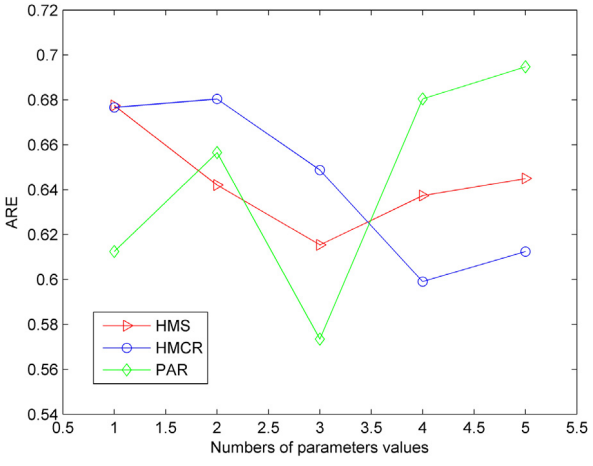


**Fig. 4.** The change trend of parameters in HHS.

analysis, the parameters in HHS are suggested as follows: HMS = 30, HMCR = 0.9, PAR = 0.2.

#### 4.2.2. Comparisons of HS, SGA, SGA+NEH, HGA and HHS for makespan criterion

To show the performance of HHS, HHS is compared with algorithms associated with GA (Zheng and Wang, 2003) and the standard HS.

**Table 9**
The corresponding values of different parameters in HHS.

| Parameters | Values | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| HMS(1) | 10 | 20 | 30 | 40 | 50 |
| HMCR(2) | 0.8 | 0.85 | 0.9 | 0.95 | 1 |
| PAR(3) | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |

We accept the results of previous papers and do not implement these algorithms by ourselves. The experimental results are reported in Table 12. From Table 12, it can be seen that in terms of BRE and ARE makespan, HHS can yield better solutions than the compared algorithms except for instance Rec05. For the Rec05, HGA can obtain optimal makespan. Fortunately, BRE makespan yielded by HHS outperforms the HGA. Meanwhile, Means plot for the above algorithms with 95% LSD intervals is presented in Fig. 5 to exhibit the stability of HHS. From Fig. 5, the interval of HHS is narrower than that of all others, which manifests that HHS has good robustness. So HHS is a good alternative algorithm for dealing with PFFSP.

#### 4.2.3. Comparisons of DE, HDE, L-HDE and HHS for makespan criterion

In order to further verify the efficiency of HHS, HHS is compared with other three state-of-art algorithms associated with DE, these algorithms include HDE (Qian et al., 2008), DE and L-HDE (Liu et al., 2014), which have already achieved excellent performance. As a matter of convenience, we accept the results and all the results of compared

**Table 10**
Parameter combinations and ARE of Rec19 instance.

| Num | Combinations | | | ARE |
|---|---|---|---|---|
| | HMS | HMCR | PAR | |
| 1 | 10(1) | 0.8(1) | 0.1(1) | 0.6928 |
| 2 | 10(1) | 0.85(2) | 0.2(2) | 0.7836 |
| 3 | 10(1) | 0.9(3) | 0.3(3) | 0.5256 |
| 4 | 10(1) | 0.95(4) | 0.4(4) | 0.6593 |
| 5 | 10(1) | 1(5) | 0.5(5) | 0.7262 |
| 6 | 20(2) | 0.8(1) | 0.2(2) | 0.5877 |
| 7 | 20(2) | 0.85(2) | 0.3(3) | 0.6641 |
| 8 | 20(2) | 0.9(3) | 0.4(4) | 0.6498 |
| 9 | 20(2) | 0.95(4) | 0.5(5) | 0.6450 |
| 10 | 20(2) | 1(5) | 0.1(1) | 0.6641 |
| 11 | 30(3) | 0.8(1) | 0.3(3) | 0.5638 |
| 12 | 30(3) | 0.85(2) | 0.4(4) | 0.7406 |
| 13 | 30(3) | 0.9(3) | 0.5(5) | 0.6785 |
| 14 | 30(3) | 0.95(4) | 0.1(1) | 0.5399 |
| 15 | 30(3) | 1(5) | 0.2(2) | 0.5542 |
| 16 | 40(4) | 0.8(1) | 0.4(4) | 0.7788 |
| 17 | 40(4) | 0.85(2) | 0.5(5) | 0.6641 |
| 18 | 40(4) | 0.9(3) | 0.1(1) | 0.6163 |
| 19 | 40(4) | 0.95(4) | 0.2(2) | 0.5829 |
| 20 | 40(4) | 1(5) | 0.3(3) | 0.5447 |
| 21 | 50(5) | 0.8(1) | 0.5(5) | 0.7597 |
| 22 | 50(5) | 0.85(2) | 0.1(1) | 0.5495 |
| 23 | 50(5) | 0.9(3) | 0.2(2) | 0.7740 |
| 24 | 50(5) | 0.95(4) | 0.3(3) | 0.5686 |
| 25 | 50(5) | 1(5) | 0.4(4) | 0.5733 |

algorithms except for HHS are directly taken from literatures (Liu et al., 2014; Qian et al., 2008). And the statistical results are listed in Table 13. It can be easily seen from Table 13 that the HHS wins the first place according to the overall mean of ARE, BRE and WRE. Furthermore, take the BRE and ARE as indicators, HHS can produce 24 (24/29 = 82.76%) best BRE values and 21 (21/29 = 72.41%) best ARE values, the results are summarized as HHS/29/82.76%/72.41%. the statistical results of the other compared approaches are as follows: DE/29/31.03%/10.35%, HDE/29/68.97%/31.03% and L-HDE/29/75.86%/62.07%. From the above statistics, the superiority of HHS is very obvious. In addition, Means plot for the above algorithms with 95% LSD intervals are also provided in Fig. 6 to show the good performance of HHS.

#### 4.2.4. Comparisons of standard PSOVNS, PSOMA, LDPSO and HHS for makespan criterion

For Table 14, HHS is further also made a comparison with other state-of-the-art approaches associated with PSO including PSOVNS (Tasgetiren et al., 2007), PSOMA (Liu et al., 2007) and LDPSO (Zhao et al., 2014b). These algorithms have also already gained prefect performance. For the sake of brevity, we accept the results and do not implement them by ourselves. Apart from the HHS, the rest results are all taken directly from the literatures (Liu et al., 2007; Tasgetiren et al., 2007; Zhao et al., 2014b). From Table 14, it can be seen that the results achieved by HHS is superior to those of the other three approaches on almost all the benchmarks except for instances Rec05 and Rec37. For the Rec05 with indicator of BRE, LDPSO is the winner. For Rec37 with indicator of BRE, PSOMA is a winner. Luckily, for Rec05 and Rec37 with indicator of ARE, HHS is evidently better than LDPSO and PSOMA. For the computational time, the $t_{avg}$ of HHS is smaller than that of LDPSO. Besides, Fig. 7 shows means plot for the above algorithms with 95% LSD intervals, which illustrates that the HHS can generate statistically better results than the others.

With different indicators, HHS is compared with a variety of meta-heuristics. From the above comparisons, it can be concluded than HHS outperforms the other approaches. This also indicates that hybridization with VNS is effective and efficient. And then, to illustrate the convergence speed of HHS more clearly, some representative cases of standard HS and HHS are shown in Fig. 8 and indicated by a letter (a through f). With regard to these Figures, we find that the convergence speed of

**Table 11**
Parameters rank and response values.

| Parameters values number | HMS | HMCR | PAR |
|---|---|---|---|
| 1 | 0.6775 | 0.6766 | 0.6125 |
| 2 | 0.6421 | 0.6804 | 0.6565 |
| 3 | 0.6154 | 0.6488 | 0.5734 |
| 4 | 0.6374 | 0.5991 | 0.6804 |
| 5 | 0.6450 | 0.6125 | 0.6947 |
| Response values | 0.0223 | 0.0368 | 0.0501 |
| Rank | 3 | 2 | 1 |

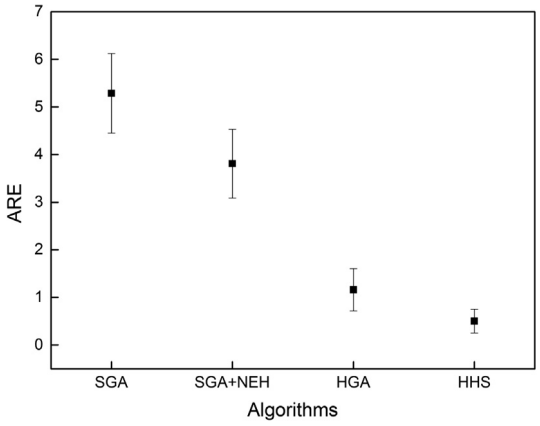


**Fig. 5.** Means plot for SGA, SGA+NEH, HGA and HHS with 95% LSD intervals.

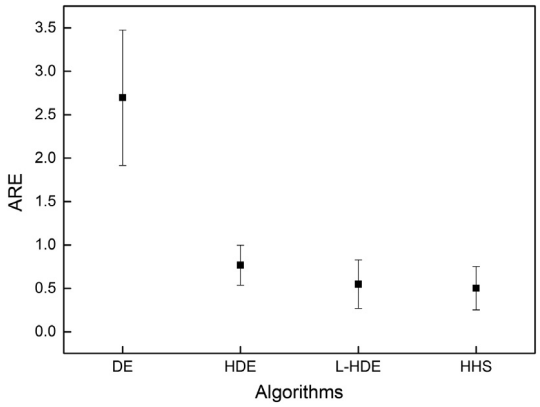


**Fig. 6.** Means plot for DE, HDE, L-HDE and HHS with 95% LSD intervals.

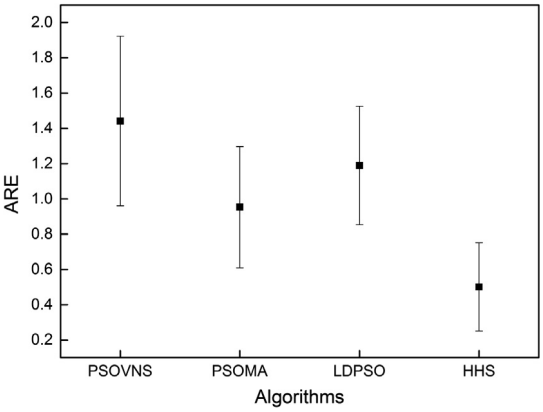


**Fig. 7.** Means plot for PSOVNS, PSOMA, LDPSO and HHS with 95% LSD intervals.

**Table 12**
Compared results of HS, SGA, SGA+NEH, HGA and HHS for makespan criterion.

| Problem | HS | | SGA | | SGA + NEH | | HGA | | HHS | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BRE | ARE | BRE | ARE | BRE | ARE | BRE | ARE | BRE | ARE |
| Car1 | **0.0000** | **0.0000** | **0** | 0.27 | **0** | **0** | **0** | **0** | **0.0000** | **0.0000** |
| Car2 | **0.0000** | 1.5499 | **0** | 4.07 | 2.93 | 2.93 | **0** | **0** | **0.0000** | **0.0000** |
| Car3 | 0.8616 | 1.9520 | 1.09 | 2.95 | 0.82 | 1.21 | **0** | **0** | **0.0000** | **0.0000** |
| Car4 | **0.0000** | 0.6173 | **0** | 2.36 | **0** | 0.07 | **0** | **0** | **0.0000** | **0.0000** |
| Car5 | 0.3756 | 1.1287 | **0** | 1.46 | **0** | 1.14 | **0** | **0** | **0.0000** | **0.0000** |
| Car6 | **0.0000** | 1.0770 | **0** | 1.86 | **0** | 2.82 | **0** | 0.04 | **0.0000** | **0.0000** |
| Car7 | **0.0000** | 0.5898 | **0** | 1.57 | **0** | 1.36 | **0** | **0** | **0.0000** | **0.0000** |
| Car8 | **0.0000** | 0.5961 | **0** | 2.59 | **0** | 0.03 | **0** | **0** | **0.0000** | **0.0000** |
| Rec01 | 4.6512 | 6.7629 | 2.81 | 6.96 | 2.25 | 6.13 | **0** | 0.14 | **0.0000** | **0.1123** |
| Rec03 | 1.8034 | 4.4304 | 1.89 | 4.45 | 1.26 | 4.27 | **0** | 0.09 | **0.0000** | **0.0000** |
| Rec05 | 1.6103 | 2.6624 | 1.93 | 3.82 | 2.33 | 2.90 | **0** | 0.29 | 0.2415 | **0.2415** |
| Rec07 | 3.3844 | 5.6407 | 1.15 | 5.31 | 3.38 | 5.27 | **0** | 0.69 | **0.0000** | **0.0000** |
| Rec09 | 5.9206 | 7.5689 | 3.12 | 4.73 | 0.39 | 2.13 | **0** | 0.64 | **0.0000** | **0.0000** |
| Rec11 | 5.5905 | 8.7212 | 3.91 | 7.39 | 1.19 | 3.66 | **0** | 1.10 | **0.0000** | **0.0000** |
| Rec13 | 5.2332 | 6.8014 | 3.68 | 5.97 | 1.92 | 4.41 | 0.36 | 1.68 | **0.0000** | **0.2798** |
| Rec15 | 4.5128 | 5.8530 | 2.21 | 4.29 | 2.87 | 4.02 | 0.56 | 1.12 | **0.0000** | **0.2974** |
| Rec17 | 7.5184 | 8.8258 | 3.15 | 6.08 | 2.16 | 4.02 | 0.95 | 2.32 | **0.0000** | **0.1507** |
| Rec19 | 8.0745 | 10.4921 | 4.01 | 6.07 | 2.05 | 4.35 | 0.62 | 1.32 | **0.2867** | **0.6179** |
| Rec21 | 7.6847 | 10.0545 | 3.42 | 6.07 | 3.52 | 3.58 | 1.44 | 1.57 | **0.4462** | **1.2494** |
| Rec23 | 8.7519 | 10.7310 | 3.83 | 7.46 | 3.63 | 5.12 | **0.40** | 0.87 | 0.4475 | **0.4840** |
| Rec25 | 7.6403 | 10.0040 | 4.42 | 7.20 | 3.14 | 4.89 | 1.27 | 2.54 | **0.4377** | **0.8542** |
| Rec27 | 9.4817 | 10.9875 | 4.93 | 6.85 | 3.16 | 5.12 | 1.10 | 1.83 | **0.4635** | **0.9496** |
| Rec29 | 9.9257 | 13.1526 | 6.21 | 8.48 | 3.32 | 4.93 | 1.40 | 2.70 | **0.1312** | **0.8191** |
| Rec31 | 10.9360 | 12.2496 | 6.17 | 8.02 | 5.94 | 6.66 | 0.43 | 1.34 | **0.4269** | **1.0082** |
| Rec33 | 7.3539 | 8.5014 | 3.08 | 5.12 | 2.70 | 3.38 | **0** | 0.78 | **0.0000** | **0.2119** |
| Rec35 | 3.5093 | 5.1775 | 1.46 | 3.30 | 1.89 | 2.58 | **0** | **0** | **0.0000** | **0.0000** |
| Rec37 | 15.2090 | 16.2553 | 7.89 | 10.07 | 7.14 | 7.94 | 3.75 | 4.90 | **2.5651** | **2.8297** |
| Rec39 | 13.4657 | 14.6006 | 7.32 | 8.51 | 6.25 | 7.09 | 2.20 | 2.79 | **1.0615** | **1.6316** |
| Rec41 | 15.6855 | 16.9382 | 8.51 | 10.03 | 7.49 | 8.49 | 3.64 | 4.92 | **2.2177** | **2.8004** |
| Average | 5.4890 | 7.0318 | 2.97 | 5.29 | 2.47 | 3.81 | 0.63 | 1.16 | **0.3009** | **0.5013** |

**Table 13**
Compared results of DE, HDE, L-HDE and HHS for the makespan criterion.

| Problem | DE | | | HDE | | | L-HDE | | | HHS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BRE | ARE | WRE | BRE | ARE | Std | BRE | ARE | WRE | BRE | ARE | WRE | Std |
| Car1 | **0** | **0** | **0** | **0.000** | **0.000** | **0.000** | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Car2 | **0** | **0** | **0** | **0.000** | **0.000** | **0.000** | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Car3 | **0** | 0.0369 | 0.7385 | **0.000** | 0.536 | 44.352 | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Car4 | **0** | **0** | **0** | **0.000** | **0.000** | **0.000** | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Car5 | **0** | 0.5285 | 1.3083 | **0.000** | 0.593 | 49.537 | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Car6 | **0** | 0.3821 | 0.7643 | **0.000** | 0.153 | 27.406 | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Car7 | **0** | 0.5341 | 2.8680 | **0.000** | 0.448 | 60.219 | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Car8 | **0** | 0.0645 | 0.6455 | **0.000** | **0.000** | **0.000** | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Rec01 | 0.1604 | 0.2566 | 0.6415 | **0.000** | 0.152 | **0.447** | **0** | **0** | **0** | **0.0000** | 0.1123 | 0.1604 | 0.9155 |
| Rec03 | 0.1803 | 0.2976 | 0.9214 | **0.000** | 0.153 | 1.252 | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Rec05 | 0.2415 | 0.2657 | 0.4026 | **0.242** | 0.386 | 3.327 | **0.2415** | **0.2415** | **0.2415** | **0.2415** | **0.2415** | **0.2415** | **0.0000** |
| Rec07 | 0.7024 | 1.1047 | 1.1494 | **0.000** | 0.920 | 7.589 | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Rec09 | 0.4554 | 2.4919 | 5.8556 | **0.000** | 0.273 | 11.593 | 0.0260 | 0.0260 | 0.0260 | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Rec11 | 1.6073 | 3.3403 | 6.1495 | **0.000** | **0.000** | **0.000** | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Rec13 | 1.1399 | 2.5648 | 3.9896 | 0.259 | 0.705 | 8.566 | **0** | **0.2746** | 0.7772 | **0.0000** | 0.2798 | **0.7254** | **2.9228** |
| Rec15 | 1.4872 | 2.3026 | 4.8205 | 0.368 | 1.309 | 11.874 | **0** | 0.5231 | **1.1795** | **0.0000** | **0.2974** | **1.1795** | **7.3114** |
| Rec17 | 3.6278 | 5.6046 | 7.9390 | **0.000** | 0.5573 | **1.8402** | **0** | 0.3628 | **0.9464** | **0.0000** | **0.1507** | 1.1567 | 6.0222 |
| Rec19 | 1.6722 | 3.5882 | 5.1123 | **0.287** | 0.908 | 9.104 | **0.2867** | 0.7023 | 1.2422 | **0.2867** | **0.6179** | **0.9078** | **5.4178** |
| Rec21 | 1.6361 | 4.1200 | 7.6847 | **0.198** | 1.284 | 9.171 | 0.6445 | 1.2791 | **1.4378** | 0.4462 | **1.2494** | **1.4378** | **8.0552** |
| Rec23 | 2.5361 | 4.2367 | 6.0169 | 0.497 | 0.696 | 10.625 | **0.3481** | **0.4276** | **0.4973** | 0.4475 | 0.4840 | **0.4973** | **0.4577** |
| Rec25 | 2.7059 | 4.4489 | 6.4465 | 0.676 | 1.429 | 16.489 | 0.5571 | 1.0824 | 1.6315 | **0.4377** | **0.8542** | **1.2734** | **7.2098** |
| Rec27 | 2.0649 | 4.7830 | 7.4168 | 0.843 | 1.197 | **4.600** | **0.2528** | **0.8512** | **1.2221** | 0.4635 | 0.9496 | 1.3485 | 4.6270 |
| Rec29 | 5.3782 | 7.1404 | 9.9694 | 0.525 | 1.299 | 13.342 | 0.8308 | 1.0494 | 1.4429 | **0.1312** | **0.8191** | **1.5304** | **9.0461** |
| Rec31 | 1.8391 | 5.5107 | 8.1445 | 0.427 | 1.192 | 13.107 | **0.4269** | **0.6437** | **0.9195** | **0.4269** | 1.0082 | 1.3793 | **10.1147** |
| Rec33 | 1.4451 | 3.5196 | 4.4637 | 0.353 | 0.787 | **4.743** | **0** | 0.2441 | **0.8349** | **0.0000** | **0.2119** | **0.8349** | 7.7345 |
| Rec35 | **0** | 1.4831 | 2.8380 | **0.000** | **0.000** | **0.000** | **0** | **0** | **0** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Rec37 | 5.7160 | 6.8350 | 7.8772 | **1.697** | **2.632** | 33.410 | 2.5651 | 3.0014 | 3.5548 | 2.5651 | 2.8297 | **3.2519** | **9.4398** |
| Rec39 | 4.0495 | 5.4767 | 6.9196 | 1.278 | **1.543** | **7.863** | 1.7299 | 1.8321 | **2.0051** | **1.0615** | 1.6316 | 2.2017 | 9.7692 |
| Rec41 | 6.0081 | 6.9073 | 7.3992 | **1.714** | **2.615** | 36.387 | 2.6613 | 3.3508 | 3.7702 | 2.2177 | 2.8004 | **3.5685** | **15.1631** |
| Average | 1.5398 | 2.6951 | 4.0787 | 0.325 | 0.766 | 13.792 | 0.3636 | 0.5480 | 0.7573 | **0.3009** | **0.5013** | **0.7481** | **3.5933** |

HHS is faster than that of HS. It is worth mentioning that the starting point of HS and HHS is not on the same point, which is caused by the NEH initialization. Meanwhile, some of optimal schedule Gantt charts are also offered in Figs. 9–12.

*4.2.5. Comparisons of standard HS, ATPSO, HPSO, L-HDE and HHS for minimizing makespan on Taillard set*

In this section, HHS algorithm is compared on different scales based on Taillard set. The upper bounds (available from http://mistic.

**Table 14**
Compared results of PSOVNS, PSOMA, LDPSO and HHS for makspan criterion.

| Problem | PSOVNS | | | PSOMA | | | LDPSO | | | | HHS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BRE | ARE | WRE | BRE | ARE | WRE | BRE | ARE | WRE | $t_{avg}$ (s) | BRE | ARE | WRE | $t_{avg}$ (s) |
| Car1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | 0.17 | **0.0000** | **0.0000** | **0.0000** | 0.0061 |
| Car2 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | 0.27 | **0.0000** | **0.0000** | **0.0000** | 0.0347 |
| Car3 | **0** | 0.420 | 1.189 | **0** | **0** | **0** | **0** | 0.49 | 0.94 | 0.24 | **0.0000** | **0.0000** | **0.0000** | 0.1392 |
| Car4 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | 0.23 | **0.0000** | **0.0000** | **0.0000** | 0.0306 |
| Car5 | **0** | 0.039 | 0.389 | **0** | 0.018 | 0.375 | **0** | 0.52 | 1.03 | 0.21 | **0.0000** | **0.0000** | **0.0000** | 0.1038 |
| Car6 | **0** | 0.076 | 0.764 | **0** | 0.114 | 0.764 | **0** | 0.55 | 0.67 | 0.14 | **0.0000** | **0.0000** | **0.0000** | 0.0187 |
| Car7 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | 0.07 | **0.0000** | **0.0000** | **0.0000** | 0.0062 |
| Car8 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | 0.01 | 0.54 | 0.21 | **0.0000** | **0.0000** | **0.0000** | 0.0046 |
| Rec01 | 0.160 | 0.168 | 0.321 | **0** | 0.144 | 0.160 | 0.11 | 0.32 | 1.75 | 1.58 | **0.0000** | **0.1123** | **0.1604** | 7.8372 |
| Rec03 | **0** | 0.158 | 0.180 | **0** | 0.189 | 0.721 | **0** | 0.22 | 0.59 | 1.48 | **0.0000** | **0.0000** | **0.0000** | 1.9824 |
| Rec05 | 0.242 | 0.249 | 0.420 | 0.242 | 0.249 | 0.402 | **0.19** | 0.29 | 0.74 | 1.14 | 0.2415 | **0.2415** | **0.2415** | 8.9621 |
| Rec07 | 0.702 | 1.095 | 1.405 | **0** | 0.986 | 1.149 | **0** | 1.55 | 3.49 | 2.47 | **0.0000** | **0.0000** | **0.0000** | 2.4632 |
| Rec09 | **0** | 0.651 | 1.366 | 0.0260 | 0.621 | 1.691 | **0** | 1.24 | 2.21 | 2.98 | **0.0000** | **0.0000** | **0.0000** | 1.7817 |
| Rec11 | 0.071 | 1.153 | 2.656 | **0** | 0.129 | 0.978 | **0** | 1.21 | 2.92 | 3.28 | **0.0000** | **0.0000** | **0.0000** | 2.0961 |
| Rec13 | 1.036 | 1.79 | 2.643 | 0.259 | 0.893 | 1.502 | 0.06 | 1.02 | 2.77 | 5.59 | **0.0000** | **0.2798** | **0.7254** | 11.9087 |
| Rec15 | 0.769 | 1.487 | 2.256 | 0.051 | 0.628 | **1.076** | 0.44 | 1.21 | 2.24 | 4.79 | **0.0000** | **0.2974** | 1.1795 | 11.1094 |
| Rec17 | 0.999 | 2.453 | 3.365 | **0** | 1.330 | 2.155 | 0.29 | 2.91 | 5.01 | 4.87 | **0.0000** | **0.1507** | **1.1567** | 7.2093 |
| Rec19 | 1.529 | 2.099 | 2.532 | 0.430 | 1.313 | 2.102 | 0.47 | 1.26 | 2.03 | 13.22 | **0.2867** | **0.6179** | **0.9078** | 27.7862 |
| Rec21 | 1.487 | 1.671 | 2.033 | 1.437 | 1.596 | 1.636 | 1.38 | 1.49 | 2.38 | 10.16 | **0.4462** | **1.2494** | **1.4378** | 27.5059 |
| Rec23 | 1.343 | 2.106 | 2.884 | 0.596 | 1.310 | 2.038 | 0.62 | 1.58 | 2.83 | 15.06 | **0.4475** | **0.4840** | **0.4973** | 27.4830 |
| Rec25 | 2.388 | 3.166 | 3.780 | 0.835 | 2.085 | 3.233 | 0.49 | 2.13 | 3.28 | 21.03 | **0.4377** | **0.8542** | **1.2734** | 33.6396 |
| Rec27 | 1.728 | 2.463 | 3.203 | 1.348 | 1.605 | 2.402 | 1.29 | 2.05 | 3.27 | 18.62 | **0.4635** | **0.9496** | **1.3485** | 33.3240 |
| Rec29 | 1.968 | 3.109 | 4.067 | 1.442 | 1.888 | 2.492 | 1.01 | 2.24 | 3.59 | 19.86 | **0.1312** | **0.8191** | **1.5304** | 32.5300 |
| Rec31 | 2.594 | 3.232 | 4.237 | 1.510 | 2.254 | 2.692 | 1.41 | 2.21 | 2.76 | 65.89 | **0.4269** | **1.0082** | **1.3793** | 97.0554 |
| Rec33 | 0.835 | 1.007 | 1.477 | **0** | 0.645 | 0.834 | 0.41 | 0.96 | 2.65 | 52.11 | **0.0000** | **0.2119** | **0.8349** | 90.1304 |
| Rec35 | **0** | 0.038 | 0.092 | **0** | **0** | **0** | **0** | 0.03 | 0.28 | 41.41 | **0.0000** | **0.0000** | **0.0000** | 4.9835 |
| Rec37 | 4.383 | 4.949 | 5.736 | **2.101** | 3.537 | 4.039 | 2.61 | 3.39 | 4.27 | 515.38 | 2.5651 | **2.8297** | **3.2519** | 451.5775 |
| Rec39 | 2.850 | 3.371 | 3.951 | 1.553 | 2.426 | 2.830 | 1.48 | 2.29 | 4.02 | 602.68 | **1.0615** | **1.6316** | **2.2017** | 454.9816 |
| Rec41 | 4.173 | 4.867 | 5.585 | 2.641 | 3.684 | 4.052 | 2.93 | 3.48 | 4.91 | 543.36 | **2.2177** | **2.8004** | **3.5685** | 469.2074 |
| Average | 1.0089 | 1.4420 | 1.9493 | 0.4981 | 0.9532 | 1.3560 | 0.52 | 1.19 | 2.11 | 67.19 | **0.3009** | **0.5013** | **0.7481** | 62.2724 |

**Table 15**
Compared results of standard HS and HHS for makespan criterion on the Taillard benchmarks.

| Problems | $n, m$ | $C^*$ | HS | | | | | HHS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | BRE | ARE | WRE | Std | $t_{avg}$ (s) | BRE | ARE | WRE | Std | $t_{avg}$ (s) |
| Ta001 | 20,5 | 1278–1278 | 1.4867 | 2.5300 | 3.9124 | 10.8934 | 0.0323 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 1.4578 |
| Ta002 | 20,5 | 1359–1359 | 0.6623 | 1.4373 | 1.7660 | 5.2217 | 0.0320 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 2.3977 |
| Ta003 | 20,5 | 1081–1081 | 4.2553 | 6.7345 | 10.7308 | 17.8053 | 0.0321 | 0.0000 | 0.0555 | 0.5550 | 1.8974 | 4.4301 |
| Ta004 | 20,5 | 1293–1293 | 4.0990 | 5.9036 | 7.5019 | 14.9316 | 0.0327 | 0.0000 | 0.0722 | 0.6187 | 2.2509 | 4.2977 |
| Ta005 | 20,5 | 1235–1235 | 2.9150 | 4.8529 | 7.9352 | 17.4006 | 0.0320 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 1.8529 |
| Ta006 | 20,5 | 1195–1195 | 3.0126 | 4.8982 | 7.3640 | 17.0665 | 0.0323 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.6377 |
| Ta007 | 20,5 | 1234–1234 | 2.0259 | 3.4360 | 5.5105 | 12.7660 | 0.0332 | 0.4052 | 0.7536 | 1.3776 | 5.7359 | 8.6260 |
| Ta008 | 20,5 | 1206–1206 | 4.3118 | 5.7822 | 7.3798 | 10.6802 | 0.0324 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.9954 |
| Ta009 | 20,5 | 1230–1230 | 4.3902 | 6.1301 | 9.2683 | 16.1635 | 0.0321 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 2.0486 |
| Ta010 | 20,5 | 1108–1108 | 2.5271 | 5.8303 | 7.6715 | 16.8938 | 0.0342 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 1.0388 |
| Average | | | 2.9686 | 4.7535 | 6.9040 | 13.9823 | 0.0325 | 0.0405 | 0.0881 | 0.2551 | 0.9884 | 2.7783 |
| Ta011 | 20,10 | 1582–1582 | 6.1947 | 7.6148 | 9.4817 | 16.4311 | 0.0362 | 0.0000 | 0.1138 | 0.2528 | 1.5492 | 10.7473 |
| Ta012 | 20,10 | 1659–1659 | 6.0277 | 7.7959 | 10.1266 | 20.1057 | 0.0354 | 0.0000 | 0.3818 | 1.3261 | 6.4328 | 9.8399 |
| Ta013 | 20,10 | 1496–1496 | 6.1497 | 7.8788 | 9.8930 | 17.1167 | 0.0341 | 0.0000 | 0.4456 | 0.8690 | 4.1861 | 10.6861 |
| Ta014 | 20,10 | 1377–1377 | 5.4466 | 9.1552 | 11.6195 | 24.1290 | 0.0350 | 0.0726 | 0.2275 | 0.7988 | 3.2704 | 10.1723 |
| Ta015 | 20,10 | 1491–1491 | 2.6828 | 4.3908 | 6.5728 | 16.5739 | 0.0345 | 0.0000 | 0.3571 | 0.6342 | 2.0166 | 10.2178 |
| Ta016 | 20,10 | 1397–1397 | 4.7960 | 7.6831 | 9.6636 | 16.4823 | 0.0355 | 0.0000 | 0.1384 | 0.7874 | 3.0582 | 6.7855 |
| Ta017 | 20,10 | 1484–1484 | 4.4474 | 6.2534 | 8.6253 | 16.2621 | 0.0345 | 0.0000 | 0.0719 | 0.5391 | 2.1202 | 5.9714 |
| Ta018 | 20,10 | 1538–1538 | 7.6073 | 9.5362 | 11.7685 | 23.6935 | 0.0348 | 0.0000 | 0.4898 | 0.9103 | 3.1137 | 10.9946 |
| Ta019 | 20,10 | 1593–1593 | 7.8622 | 10.1061 | 12.3457 | 23.4059 | 0.0363 | 0.0000 | 0.2385 | 0.9416 | 4.5544 | 11.2028 |
| Ta020 | 20,10 | 1591–1591 | 5.9711 | 7.9950 | 9.8680 | 18.6747 | 0.0339 | 0.0000 | 0.6537 | 1.1942 | 7.2585 | 10.4454 |
| Average | | | 5.7186 | 7.8409 | 9.9965 | 19.2875 | 0.0350 | 0.0073 | 0.3118 | 0.8254 | 3.7560 | 9.7063 |
| Ta021 | 20,20 | 2297–2297 | 5.0065 | 6.8582 | 8.5329 | 29.5172 | 0.0405 | 0.0000 | 0.3178 | 0.6095 | 4.8316 | 14.2480 |
| Ta022 | 20,20 | 2099–2099 | 4.5260 | 6.1617 | 7.9085 | 21.5296 | 0.0395 | 0.0000 | 0.3049 | 0.7623 | 6.3673 | 12.5184 |
| Ta023 | 20,20 | 2326–2326 | 5.1591 | 6.5835 | 7.9966 | 18.8485 | 0.0401 | 0.0860 | 0.4786 | 0.9458 | 5.4362 | 13.1754 |
| Ta024 | 20,20 | 2223–2223 | 5.8929 | 7.1675 | 9.2218 | 22.5283 | 0.0390 | 0.0000 | 0.3539 | 1.1696 | 6.4350 | 11.5501 |
| Ta025 | 20,20 | 2291–2291 | 3.9284 | 6.2156 | 7.7695 | 24.3715 | 0.0427 | 0.1309 | 0.3492 | 0.7420 | 4.6904 | 13.1267 |
| Ta026 | 20,20 | 2226–2226 | 4.9865 | 6.0108 | 7.8167 | 18.0404 | 0.0396 | 0.1348 | 0.3953 | 0.8086 | 4.7389 | 13.0798 |
| Ta027 | 20,20 | 2273–2273 | 4.2675 | 6.0155 | 7.3031 | 21.6975 | 0.0401 | 0.0000 | 0.4370 | 0.8799 | 5.2081 | 13.1826 |
| Ta028 | 20,20 | 2200–2200 | 4.5000 | 6.9545 | 9.0455 | 28.0204 | 0.0395 | 0.0000 | 0.3424 | 0.7727 | 4.6116 | 12.5737 |
| Ta029 | 20,20 | 2237–2237 | 6.4372 | 7.6620 | 10.3263 | 25.0194 | 0.0394 | 0.0000 | 0.2205 | 0.4023 | 1.7099 | 12.8120 |
| Ta030 | 20,20 | 2178–2178 | 5.4178 | 7.8849 | 10.3765 | 34.3271 | 0.0386 | 0.0000 | 0.2158 | 0.4591 | 3.7727 | 13.4189 |
| Average | | | 5.0122 | 6.7514 | 8.6297 | 24.3900 | 0.0399 | 0.0352 | 0.3415 | 0.7552 | 4.7802 | 12.9686 |
| Ta031 | 50,5 | 2724–2724 | 0.9912 | 1.6911 | 3.1571 | 16.2501 | 0.0430 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 3.3383 |
| Ta032 | 50,5 | 2834–2834 | 2.2583 | 4.0555 | 5.4340 | 24.7746 | 0.0409 | 0.0000 | 0.1082 | 0.1411 | 1.4864 | 71.7426 |
| Ta033 | 50,5 | 2621–2621 | 2.6326 | 3.6958 | 5.3033 | 18.4269 | 0.0391 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 21.3460 |
| Ta034 | 50,5 | 2751–2751 | 3.0534 | 4.0228 | 5.0164 | 16.5429 | 0.0395 | 0.0000 | 0.1042 | 0.3999 | 4.3072 | 55.3039 |

(*continued on next page*)

**Table 15** (*continued*)

| Problems | $n, m$ | $C^*$ | HS | | | | | HHS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | BRE | ARE | WRE | Std | $t_{avg}$ (s) | BRE | ARE | WRE | Std | $t_{avg}$ (s) |
| Ta035 | 50,5 | 2863–2863 | 1.2225 | 2.1819 | 3.1436 | 15.4174 | 0.0397 | 0.0000 | 0.0047 | 0.0349 | 0.3519 | 26.1275 |
| Ta036 | 50,5 | 2829–2829 | 2.6511 | 3.8600 | 5.0901 | 19.4833 | 0.0406 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 19.3259 |
| Ta037 | 50,5 | 2725–2725 | 3.4862 | 4.4110 | 5.1376 | 12.3242 | 0.0425 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 16.9340 |
| Ta038 | 50,5 | 2683–2683 | 2.7954 | 4.3235 | 5.8144 | 24.1631 | 0.0404 | 0.0000 | 0.0522 | 0.7827 | 5.4222 | 27.1938 |
| Ta039 | 50,5 | 2552–2552 | 3.2132 | 4.5533 | 5.4075 | 14.6785 | 0.0397 | 0.0000 | 0.0940 | 0.3527 | 2.6673 | 65.7320 |
| Ta040 | 50,5 | 2782–2782 | 1.4378 | 2.5186 | 3.6305 | 16.9978 | 0.0388 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 2.2557 |
| Average | | | 2.3742 | 3.5314 | 4.7135 | 17.9059 | 0.0404 | 0.0000 | 0.0363 | 0.1711 | 1.4235 | 30.9300 |
| Ta041 | 50,10 | 2991–2991 | 10.9662 | 13.3645 | 14.9448 | 33.7225 | 0.0455 | 1.2036 | 1.6784 | 1.9392 | 7.5100 | 111.9486 |
| Ta042 | 50,10 | 2867–2867 | 11.9288 | 13.4589 | 15.1378 | 26.5407 | 0.0451 | 1.5347 | 1.7300 | 1.9881 | 5.9257 | 98.0162 |
| Ta043 | 50,10 | 2839–2839 | 12.6805 | 15.1626 | 17.0130 | 35.7648 | 0.0450 | 1.0919 | 1.5123 | 2.3952 | 10.4161 | 96.3072 |
| Ta044 | 50,10 | 3063–3063 | 9.4025 | 10.8956 | 12.3735 | 29.0627 | 0.0463 | 0.0326 | 0.3178 | 1.2733 | 8.3106 | 96.3611 |
| Ta045 | 50,10 | 2976–2976 | 9.8454 | 12.2625 | 13.7769 | 35.6260 | 0.0461 | 1.1425 | 1.4830 | 2.0161 | 6.9884 | 96.5049 |
| Ta046 | 50,10 | 3006–3006 | 9.4810 | 11.2375 | 12.5083 | 28.0591 | 0.0473 | 0.0000 | 0.9093 | 1.4970 | 11.4184 | 93.8695 |
| Ta047 | 50,10 | 3093–3093 | 8.7617 | 10.1972 | 11.2512 | 26.6104 | 0.0464 | 1.0023 | 1.3859 | 2.3278 | 12.2233 | 97.4904 |
| Ta048 | 50,10 | 3037–3037 | 8.9233 | 10.1262 | 11.3599 | 26.1011 | 0.0455 | 0.2305 | 0.3732 | 0.5927 | 3.7922 | 98.2788 |
| Ta049 | 50,10 | 2897–2897 | 11.3221 | 12.2817 | 14.3942 | 22.6596 | 0.0447 | 0.1726 | 0.7203 | 1.3462 | 10.3016 | 97.7366 |
| Ta050 | 50,10 | 3065–3065 | 9.4290 | 11.0995 | 13.7684 | 41.5472 | 0.0455 | 0.8483 | 1.7401 | 2.1533 | 12.0574 | 100.9286 |
| Average | | | 10.2741 | 12.0086 | 13.6528 | 30.5694 | 0.0457 | 0.7259 | 1.1850 | 1.7529 | 8.8944 | 98.7442 |
| Ta051 | 50,20 | 3771–3850 | 11.3330 | 12.9824 | 14.4969 | 39.6337 | 0.0557 | 1.5065 | 1.9818 | 2.2078 | 9.4640 | 154.1185 |
| Ta052 | 50,20 | 3668–3704 | 12.3820 | 14.1192 | 15.2684 | 33.9197 | 0.0574 | 0.9989 | 1.8062 | 2.6998 | 20.9043 | 149.7438 |
| Ta053 | 50,20 | 3591–3640 | 13.1485 | 14.4258 | 15.7013 | 30.4299 | 0.0560 | 1.7857 | 2.3102 | 2.9670 | 9.7005 | 137.0267 |
| Ta054 | 50,20 | 3635–3723 | 11.2570 | 13.6889 | 15.4114 | 49.0531 | 0.0565 | 1.6116 | 2.0441 | 2.6054 | 12.5029 | 152.8523 |
| Ta055 | 50,20 | 3553–3611 | 11.5114 | 13.3333 | 14.7762 | 28.9198 | 0.0538 | 1.8554 | 2.5505 | 3.1570 | 14.5560 | 153.6015 |
| Ta056 | 50,20 | 3667–3681 | 12.8017 | 14.2573 | 15.1343 | 24.9962 | 0.0563 | 1.3583 | 2.0293 | 2.8797 | 16.5466 | 151.3042 |
| Ta057 | 50,20 | 3652–3704 | 13.5726 | 14.6537 | 17.1883 | 34.1477 | 0.0569 | 1.6739 | 2.2165 | 2.8348 | 15.1617 | 154.0698 |
| Ta058 | 50,20 | 3627–3691 | 12.7568 | 14.8306 | 16.9459 | 49.7960 | 0.0569 | 1.6798 | 2.4871 | 3.0073 | 14.9503 | 155.7474 |
| Ta059 | 50,20 | 3645–3743 | 11.8242 | 13.4239 | 14.9933 | 31.0448 | 0.0564 | 1.7900 | 2.3110 | 3.0190 | 13.4185 | 156.2574 |
| Ta060 | 50,20 | 3696–3756 | 12.0255 | 13.8997 | 15.5827 | 38.1853 | 0.0578 | 0.9318 | 1.8956 | 2.4760 | 16.5247 | 160.9597 |
| Average | | | 12.2613 | 13.9615 | 15.5499 | 36.0126 | 0.0564 | 1.5494 | 2.1976 | 2.8129 | 14.6342 | 153.6366 |
| Ta061 | 100,5 | 5493–5493 | 1.1287 | 1.7313 | 2.4213 | 23.5101 | 0.0788 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 19.2008 |
| Ta062 | 100,5 | 5268–5268 | 1.4047 | 2.5731 | 3.9863 | 34.6311 | 0.0634 | 0.0000 | 0.2012 | 0.3037 | 6.9498 | 324.5919 |
| Ta063 | 100,5 | 5175–5175 | 2.2802 | 3.1208 | 4.0386 | 25.345 | 0.1187 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 87.4241 |
| Ta064 | 100,5 | 5014–5014 | 1.4559 | 2.2956 | 3.6099 | 28.4917 | 0.1200 | 0.0000 | 0.0638 | 0.0798 | 1.7889 | 354.4542 |
| Ta065 | 100,5 | 5250–5250 | 1.7333 | 3.0229 | 4.3619 | 36.345 | 0.1195 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 302.3994 |
| Ta066 | 100,5 | 5135–5135 | 1.3242 | 1.8598 | 2.668 | 18.1992 | 0.0629 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 31.4278 |
| Ta067 | 100,5 | 5246–5246 | 1.5059 | 2.4476 | 3.2215 | 26.992 | 0.0637 | 0.0000 | 0.0496 | 0.2478 | 5.8138 | 171.5434 |
| Ta068 | 100,5 | 5094–5094 | 2.4342 | 3.4442 | 4.9274 | 30.2002 | 0.0633 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 58.3014 |
| Ta069 | 100,5 | 5448–5448 | 2.0558 | 2.7606 | 3.7628 | 26.8767 | 0.0624 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 110.1814 |
| Ta070 | 100,5 | 5332–5332 | 2.0105 | 2.9171 | 3.9459 | 27.2626 | 0.0624 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 359.4448 |
| Average | | | 1.7333 | 2.6173 | 3.6944 | 27.7854 | 0.0815 | 0.0000 | 0.0315 | 0.0631 | 1.4553 | 181.8969 |
| Ta071 | 100,10 | 5770–5770 | 8.1802 | 9.1213 | 10.7972 | 35.1794 | 0.0772 | 0.1906 | 0.2392 | 0.2600 | 1.6432 | 584.6198 |
| Ta072 | 100,10 | 5349–5349 | 8.8241 | 10.4730 | 12.0022 | 50.4231 | 0.0760 | 0.2430 | 0.4150 | 0.7852 | 12.7161 | 587.1172 |
| Ta073 | 100,10 | 5676–5676 | 6.3777 | 9.2319 | 9.2319 | 49.5889 | 0.0766 | 0.0529 | 0.1339 | 0.2643 | 6.3087 | 608.0361 |
| Ta074 | 100,10 | 5781–5781 | 9.6177 | 10.6625 | 11.4167 | 25.3510 | 0.0762 | 0.4940 | 0.6847 | 0.9860 | 5.0695 | 565.9454 |
| Ta075 | 100,10 | 5467–5467 | 9.2007 | 10.5616 | 11.6517 | 38.0310 | 0.0764 | 0.5853 | 0.7061 | 0.9329 | 7.7974 | 589.7986 |
| Ta076 | 100,10 | 5303–5303 | 7.6372 | 9.2061 | 10.7863 | 43.2089 | 0.0753 | 0.0943 | 0.2565 | 0.3394 | 6.1887 | 588.5200 |
| Ta077 | 100,10 | 5595–5595 | 8.4897 | 9.6631 | 10.2592 | 25.9803 | 0.0757 | 0.4826 | 0.7864 | 0.9830 | 4.4721 | 594.7009 |
| Ta078 | 100,10 | 5617–5617 | 7.4239 | 8.8544 | 10.1478 | 35.4093 | 0.0751 | 0.4095 | 0.5163 | 0.8545 | 9.6177 | 598.4920 |
| Ta079 | 100,10 | 5871–5871 | 6.1148 | 7.2194 | 8.2269 | 32.6743 | 0.1420 | 0.4088 | 0.5927 | 0.8687 | 11.3446 | 611.1043 |
| Ta080 | 100,10 | 5845–5845 | 5.3721 | 7.1232 | 9.0163 | 42.0467 | 0.0751 | 0.0513 | 0.3045 | 0.6159 | 16.9322 | 568.9509 |
| Average | | | 7.7238 | 9.2117 | 10.3536 | 37.7893 | 0.0826 | 0.3012 | 0.4635 | 0.6890 | 8.2090 | 589.7285 |
| Ta081 | 100,20 | 6106–6202 | 13.8826 | 15.5821 | 16.9784 | 45.7894 | 0.1038 | 1.8220 | 1.9768 | 2.5476 | 13.0565 | 912.4071 |
| Ta082 | 100,20 | 6183–6183 | 13.8929 | 15.5337 | 17.7745 | 63.8308 | 0.1045 | 2.1672 | 2.5586 | 2.9759 | 19.2146 | 963.3074 |
| Ta083 | 100,20 | 6252–6271 | 14.0275 | 15.4790 | 16.7466 | 49.8227 | 0.1042 | 2.1368 | 2.3473 | 2.7587 | 15.9593 | 971.9658 |
| Ta084 | 100,20 | 6254–6269 | 12.3441 | 14.7546 | 16.5494 | 59.1527 | 0.1033 | 1.6789 | 2.0179 | 2.4624 | 20.4377 | 961.4879 |
| Ta085 | 100,20 | 6262–6314 | 15.0431 | 15.8336 | 17.2309 | 36.1554 | 0.1143 | 1.9005 | 2.3662 | 2.6271 | 21.1376 | 952.2706 |
| Ta086 | 100,20 | 6302–6364 | 14.0478 | 14.9859 | 16.0434 | 38.9198 | 0.1033 | 2.1684 | 2.3067 | 2.6084 | 11.5412 | 969.4241 |
| Ta087 | 100,20 | 6184–6268 | 15.0128 | 16.4590 | 17.6771 | 44.6946 | 0.1037 | 1.9943 | 2.3421 | 2.6324 | 17.1523 | 986.5898 |
| Ta088 | 100,20 | 6315–6401 | 13.9353 | 15.5819 | 16.7786 | 48.8482 | 0.1041 | 2.3746 | 2.6433 | 2.8902 | 11.9875 | 969.3052 |
| Ta089 | 100,20 | 6204–6275 | 14.5020 | 15.6120 | 16.9402 | 40.1042 | 0.1036 | 2.0876 | 2.4797 | 2.7251 | 16.1493 | 978.2315 |
| Ta090 | 100,20 | 6404–6434 | 12.8536 | 13.9812 | 15.4492 | 45.0549 | 0.1046 | 1.7718 | 2.1169 | 2.5490 | 19.0840 | 728.9098 |
| Average | | | 13.9542 | 15.3803 | 16.8168 | 47.2373 | 0.1049 | 2.0102 | 2.3156 | 2.6777 | 16.5720 | 939.3899 |
| Ta091 | 200,10 | 10862–10862 | 5.1372 | 5.7973 | 6.9693 | 50.0306 | 0.1293 | 0.0921 | 0.1436 | 0.2117 | 6.8069 | 4016.6000 |
| Ta092 | 200,10 | 10480–10480 | 8.3492 | 9.1732 | 10.5725 | 56.646 | 0.1304 | 0.4389 | 0.4676 | 0.4866 | 1.1547 | 2962.7000 |
| Ta093 | 200,10 | 10922–10922 | 5.6034 | 6.3720 | 7.2148 | 49.0483 | 0.1304 | 0.3113 | 0.3131 | 0.3205 | 0.5774 | 2983.8000 |
| Ta094 | 200,10 | 10889–10889 | 4.2153 | 4.8379 | 5.5561 | 32.0388 | 0.1340 | 0.0367 | 0.0367 | 0.0367 | 0.0000 | 4015.3000 |
| Ta095 | 200,10 | 10524–10524 | 6.8605 | 8.4839 | 9.6161 | 64.6141 | 0.1290 | 0.1235 | 0.1235 | 0.1235 | 0.0000 | 4023.7000 |
| Ta096 | 200,10 | 10329–10329 | 6.4188 | 8.2443 | 9.2749 | 72.4906 | 0.1298 | 0.1549 | 0.2614 | 0.5422 | 1.0207 | 4045.6000 |
| Ta097 | 200,10 | 10854–10854 | 5.9333 | 7.0352 | 8.0800 | 67.5717 | 0.1282 | 0.2580 | 0.3378 | 0.4975 | 15.0111 | 4043.8000 |
| Ta098 | 200,10 | 10730–10730 | 6.5424 | 7.5909 | 8.8537 | 61.106 | 0.1302 | 0.2237 | 0.3262 | 0.6337 | 23.8607 | 3944.1000 |
| Ta099 | 200,10 | 10438–10438 | 7.3961 | 8.5538 | 9.4367 | 55.4192 | 0.1298 | 0.2587 | 0.2587 | 0.2587 | 0.0000 | 4033.5000 |
| Ta100 | 200,10 | 10675–10675 | 6.3326 | 7.3630 | 8.0656 | 53.5449 | 0.1300 | 0.1405 | 0.2098 | 0.2529 | 1.1102 | 3982.6000 |
| Average | | | 6.2789 | 7.3452 | 8.3640 | 56.2510 | 0.1301 | 0.2038 | 0.2478 | 0.3364 | 4.9542 | 3805.1700 |
| Ta101 | 200,20 | 11152–11195 | 12.8629 | 14.2117 | 15.1675 | 61.7942 | 0.1836 | 1.8580 | 1.9533 | 2.0009 | 9.2376 | 7212.1000 |
| Ta102 | 200,20 | 11143–11203 | 14.3176 | 15.7846 | 16.8794 | 72.8107 | 0.1821 | 1.7852 | 1.9370 | 2.1869 | 17.7341 | 7311.7000 |
| Ta103 | 200,20 | 11281–11281 | 13.9349 | 15.3196 | 16.3549 | 69.9501 | 0.1861 | 2.1984 | 2.3461 | 2.4289 | 14.4684 | 7479.3000 |

(*continued on next page*)

**Table 15** (*continued*)

| Problems | $n, m$ | $C^*$ | HS | | | | | HHS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | BRE | ARE | WRE | Std | $t_{avg}$ (s) | BRE | ARE | WRE | Std | $t_{avg}$ (s) |
| Ta104 | 200,20 | 11275–11275 | 13.6674 | 14.5384 | 15.5033 | 57.8288 | 0.1789 | 2.0843 | 2.2049 | 2.3947 | 15.4855 | 7288.6000 |
| Ta105 | 200,20 | 11259–11259 | 13.4115 | 14.1616 | 15.3921 | 66.8576 | 0.1821 | 1.3589 | 1.3944 | 1.4388 | 4.5826 | 7112.2000 |
| Ta106 | 200,20 | 11176–11176 | 13.4306 | 14.8904 | 15.9628 | 65.5313 | 0.1799 | 1.1095 | 1.3117 | 1.5211 | 16.8760 | 7005.9000 |
| Ta107 | 200,20 | 11360–11360 | 13.4947 | 14.7504 | 15.8451 | 74.3040 | 0.1909 | 1.7430 | 1.9542 | 2.2359 | 28.8444 | 7527.8000 |
| Ta108 | 200,20 | 11334–11334 | 13.0757 | 14.1084 | 15.2285 | 67.3400 | 0.1807 | 1.2882 | 1.4240 | 1.5087 | 10.2859 | 7302.1000 |
| Ta109 | 200,20 | 11192–11192 | 14.0815 | 15.0071 | 15.7881 | 60.7509 | 0.1800 | 2.2069 | 2.2218 | 2.2516 | 2.8868 | 6958.7000 |
| Ta110 | 200,20 | 11284–11284 | 13.3995 | 14.5764 | 15.5707 | 60.6974 | 0.1802 | 1.7724 | 1.9213 | 2.1801 | 18.7270 | 7205.2000 |
| Average | | | 13.5676 | 14.7349 | 15.7692 | 65.7865 | 0.1825 | 1.7405 | 1.8669 | 2.0148 | 13.9128 | 7240.3600 |
| Ta111 | 500,20 | 26040–26040 | 10.8756 | 11.567 | 12.0161 | 78.1688 | 0.4369 | 0.9409 | 1.0853 | 1.1329 | 21.1021 | 10001 |
| Ta112 | 500,20 | 26500–26520 | 10.6410 | 11.2242 | 12.1719 | 94.0153 | 0.4368 | 0.9804 | 1.0354 | 1.1312 | 15.1921 | 10330 |
| Ta113 | 500,20 | 26371–26371 | 10.0906 | 10.9031 | 11.4368 | 82.7144 | 0.4409 | 1.1717 | 1.1945 | 1.2286 | 6.1644 | 10279 |
| Ta114 | 500,20 | 26456–26456 | 9.5252 | 10.9202 | 11.5286 | 116.6833 | 0.4404 | 0.9223 | 1.0115 | 1.2625 | 30.1615 | 10187 |
| Ta115 | 500,20 | 26334–26334 | 10.3706 | 10.8806 | 11.5250 | 86.7702 | 0.4336 | 1.2000 | 1.2455 | 1.3898 | 21.3190 | 10182 |
| Ta116 | 500,20 | 26469–26477 | 10.8434 | 11.2526 | 11.6969 | 61.8260 | 0.4322 | 1.0877 | 1.1483 | 1.3088 | 60.1041 | 10996 |
| Ta117 | 500,20 | 26389–26389 | 9.7237 | 10.3816 | 11.1334 | 104.2322 | 0.4335 | 1.1406 | 1.2505 | 1.3680 | 25.5930 | 11029 |
| Ta118 | 500,20 | 26560–26560 | 10.0489 | 10.8074 | 11.4044 | 87.4766 | 0.4302 | 0.9789 | 1.1498 | 1.2801 | 30.7539 | 10227 |
| Ta119 | 500,20 | 26005–26005 | 10.7287 | 11.7379 | 12.3784 | 101.773 | 0.4314 | 1.0190 | 1.1504 | 1.3267 | 26.3470 | 10073 |
| Ta120 | 500,20 | 26457–26457 | 10.0918 | 10.9207 | 11.4336 | 89.3238 | 0.4298 | 1.1075 | 1.2086 | 1.3456 | 33.0807 | 10194 |
| Average | | | 10.2940 | 11.0595 | 11.6725 | 90.2984 | 0.4346 | 1.0549 | 1.1480 | 1.2774 | 26.9818 | 10349.8 |

**Table 16**
Compared results of HPSO, L-HEDE and HHS for makespan criterion on the Taillard benchmarks.

| Problem | n,m | $C^*$ | HPSO | | | | L-HDE | | | | HHS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Avg | Max | ARE | Min | Avg | Max | ARE | Min | Avg | Max | ARE |
| Ta001 | 20,5 | 1278–1278 | **1278** | **1278** | **1278** | **0.00** | **1278** | **1278** | **1278** | **0.00** | **1278** | **1278** | **1278** | **0.00** |
| Ta011 | 20,10 | 1582–1582 | **1582** | 1587.3 | 1596 | 0.34 | **1582** | 1585.3 | 1592 | 0.21 | **1582** | **1583.8** | **1586** | **0.11** |
| Ta021 | 20,20 | 2297–2297 | **2297** | 2307.0 | 2315 | 0.44 | **2297** | 2304.4 | 2313 | 0.32 | **2297** | **2304.3** | **2311** | 0.32 |
| Ta031 | 50,5 | 2724–2724 | **2724** | **2724** | **2724** | **0.00** | **2724** | **2724** | **2724** | **0.00** | **2724** | **2724** | **2724** | **0.00** |
| Ta041 | 50,10 | 2991–2991 | 3034 | 3053.6 | 3063 | 2.09 | **3025** | **3039.9** | 3052 | **1.63** | 3027 | 3041.2 | **3049** | 1.68 |
| Ta051 | 50,20 | 3771–3850 | 3923 | 3944.6 | 3963 | 2.46 | 3917 | 3933 | 3951 | 2.16 | **3908** | **3926.3** | **3935** | **1.98** |
| Ta061 | 100,5 | 5493–5493 | **5493** | **5493** | **5493** | **0.00** | **5493** | **5493** | **5493** | **0.00** | **5493** | **5493** | **5493** | **0.00** |
| Average | | | | | | 0.76 | | | | 0.62 | | | | **0.58** |

**Table 17**
Compared results of ATPPSO, L-HDE and HHS for makespan criterion on the Taillard benchmarks.

| Problem | n,m | $C^*$ | ATPPSO | | | | L-HDE | | | | HHS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Min | Avg | Max | ARE | Min | Avg | Max | ARE | Min | Avg | Max | ARE |
| Ta005 | 20,5 | 1235–1235 | **1235** | 1241.9 | 1250 | 0.55 | **1235** | **1235** | **1235** | **0.00** | **1235** | **1235** | **1235** | **0.00** |
| Ta010 | 20,5 | 1108–1108 | **1108** | 1109.5 | 1120 | 0.13 | **1108** | **1108** | **1108** | **0.00** | **1108** | **1108** | **1108** | **0.00** |
| Ta020 | 20,10 | 1591–1591 | **1591** | 1607.3 | 1618 | 1.02 | **1591** | **1598.4** | **1604** | **0.47** | **1591** | 1601.4 | 1610 | 0.65 |
| Ta030 | 20,20 | 2178–2178 | **2178** | 2185.5 | 2206 | 0.35 | **2178** | 2185.0 | 2192 | 0.32 | **2178** | **2182.7** | **2188** | **0.22** |
| Ta050 | 50,10 | 3065–3065 | 3120 | 3150 | 3175 | 2.77 | 3099 | 3116.5 | **3131** | 1.68 | **3092** | **3111.2** | 3132 | **1.51** |
| Ta060 | 50,20 | 3696–3756 | 3864 | 3897.7 | 3926 | 3.77 | 3804 | **3821.1** | **3836** | **1.73** | **3791** | 3827.2 | 3849 | 1.90 |
| Ta070 | 100,5 | 5322–5322 | 5330 | 5341.0 | 5355 | 0.35 | **5322** | **5322** | **5322** | **0.00** | **5322** | **5322** | **5322** | **0.00** |
| Ta080 | 100,10 | 5845–5845 | 5930 | 5913.3 | 5979 | 1.16 | **5848** | 5881 | 5903 | 0.61 | **5848** | **5862.8** | **5881** | **0.30** |
| Average | | | | | | 1.26 | | | | 0.60 | | | | **0.57** |

**Table 18**
Compared results of standard HS, SAOP, ILS, NEGAVNS, ACA (best), NACA, FF, NEHI, HLBS and HHS for minimizing makespan on Taillard set.

| Test problem | HHS | HHS(best) | HS | SAOP | ILS | $NEGA_{VNS}$ | ACA(best) | NACA | FF | NEHI | HLBS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 × 5 | 0.09 | 0.04 | 4.75 | 1.05 | 0.33 | **0.00** | 0.39 | 0.13 | 2.29 | 1.96 | 0.02 |
| 20 × 10 | 0.31 | 0.01 | 7.84 | 2.60 | 0.52 | 0.01 | 0.83 | 0.39 | 4.15 | 3.28 | **0.00** |
| 20 × 20 | 0.34 | 0.04 | 6.75 | 2.06 | 0.28 | 0.02 | 0.94 | 0.32 | 3.31 | 3.04 | **0.01** |
| 50 × 5 | 0.04 | 0.00 | 3.53 | 0.34 | 0.18 | **0.00** | 0.09 | 0.07 | 0.92 | 0.41 | **0.00** |
| 50 × 10 | 1.19 | 0.73 | 12.01 | 3.50 | 1.45 | 0.82 | 1.24 | 0.74 | 5.15 | 3.33 | **0.50** |
| 50 × 20 | 2.20 | 1.55 | 13.96 | 4.66 | 2.05 | 1.08 | 1.99 | 1.81 | 6.21 | 4.39 | **0.59** |
| 100 × 5 | 0.03 | **0.00** | 2.62 | 0.30 | 0.16 | **0.00** | 0.07 | 0.08 | 0.38 | 0.31 | 0.03 |
| 100 × 10 | 0.46 | 0.30 | 9.21 | 1.34 | 0.64 | 0.14 | 1.06 | 0.56 | 2.18 | 1.51 | **0.12** |
| 100 × 20 | 2.32 | 2.01 | 15.38 | 4.49 | 2.42 | 1.40 | 1.83 | 1.29 | 5.02 | 4.97 | **0.83** |
| 200 × 10 | 0.25 | 0.20 | 7.35 | 0.94 | 0.50 | 0.16 | 0.43 | 0.34 | 0.98 | 0.92 | **0.06** |
| 200 × 20 | 1.87 | 1.74 | 14.73 | 3.67 | 2.07 | 1.25 | 1.24 | **0.74** | 4.04 | 3.77 | 0.95 |
| 500 × 20 | 1.15 | 1.05 | 11.06 | 2.20 | 1.20 | 0.71 | 1.44 | 0.52 | 1.78 | 1.69 | **0.45** |
| Average | 0.85 | 0.64 | 9.10 | 2.26 | 0.98 | 0.47 | 0.96 | 0.58 | 3.03 | 2.47 | **0.30** |

heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/flowshop.dir/best_lb_up.txt) for these benchmarks are provided in Table 15, and the generations of 500 × 20 instances are set to 50. For the sake of convenience and reliability, we do not implement the compared algorithm, except for HS and HHS, the rest of results reported in this section come from the literatures (Kuo et al., 2009; Liu et al., 2014; Zhang and Sun, 2009). From Table 15, it can be observed that results gained by HHS outperform those of standard HS on all the benchmarks, which verifies the efficiency and effectiveness of the HHS and HHS is very suitable for PFSSP. To intuitively illustrate the experimental results,

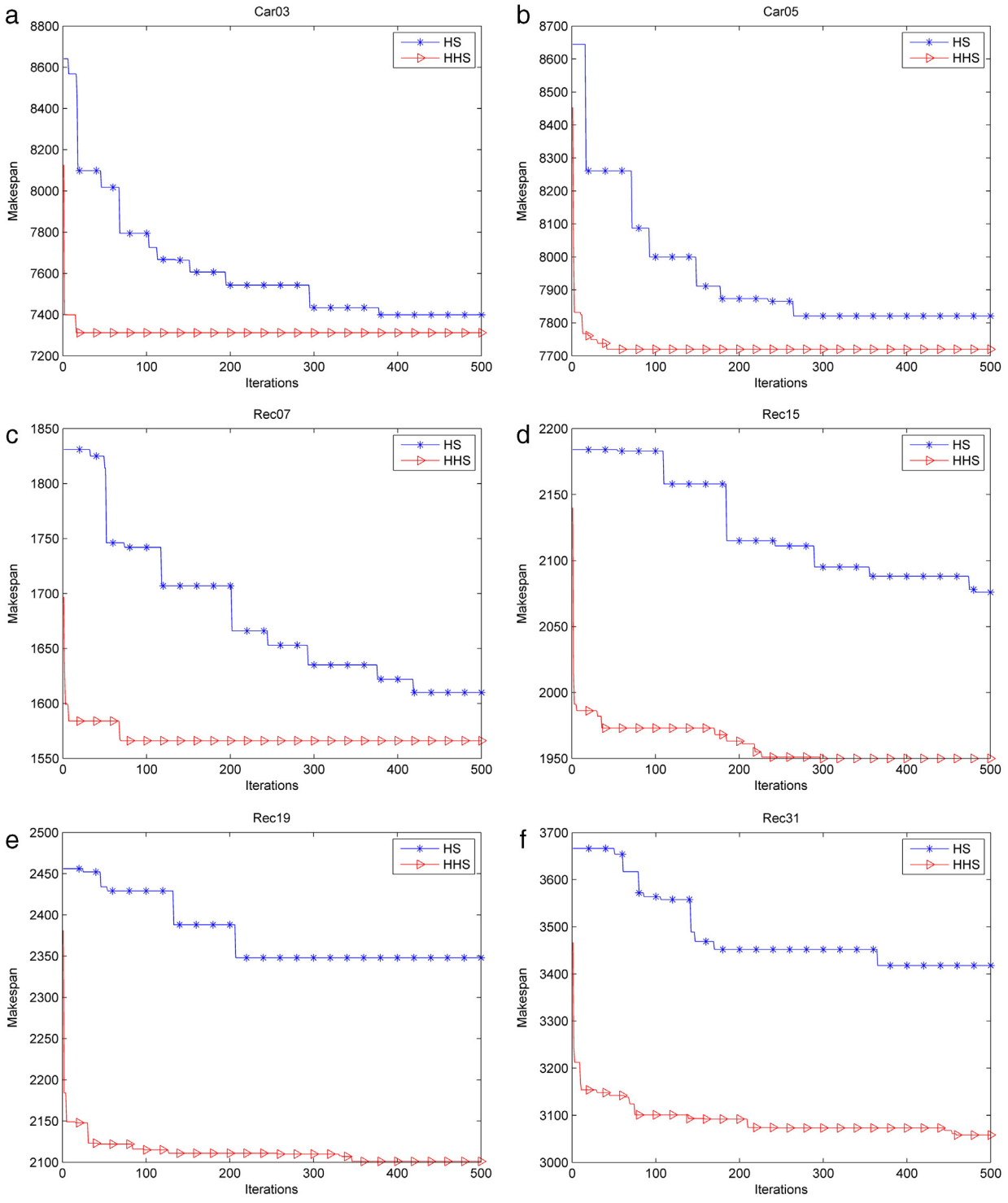


**Fig. 8.** Convergence curves of some typical benchmarks.

the convergence curves of Ta010, Ta030, Ta050 and Ta070, which are very difficult to obtain optimal values, are shown in Fig. 13 indicated by a letter ($g$ through $k$). With regard to these Figures, we find that the convergence speed of HHS is faster than that of HS. It is worth mentioning that the starting points of HS and HHS are not on the same point, which is caused by the NEH initialization.

With the maximal makspan (Max), average makespan (Avg), minimal makespan (Min) and ARE, the statistical performances of HPSO (Kuo et al., 2009), L-HDE (Liu et al., 2014) and HHS are listed in Table 16. From Table 16, we can find that HHS generates better solutions than others except for Ta041. For the Ta041, L-HDE is a winner. However, it is clear that the Min, Avg, Max and ARE yielded by HHS are superior to those of HPSO and L-HDE for the rest benchmarks. This shows the best performance of HHS and effectiveness of hybridization with VNS. In order to more clearly show the results, a histogram with indicators of BRE and ARE is provided in Fig. 14. Fig. 14 consists of two parts, the left part is for ARE comparison, and the right part is for BRE comparison. From Fig. 14, HHS evidently outperforms the other algorithms.

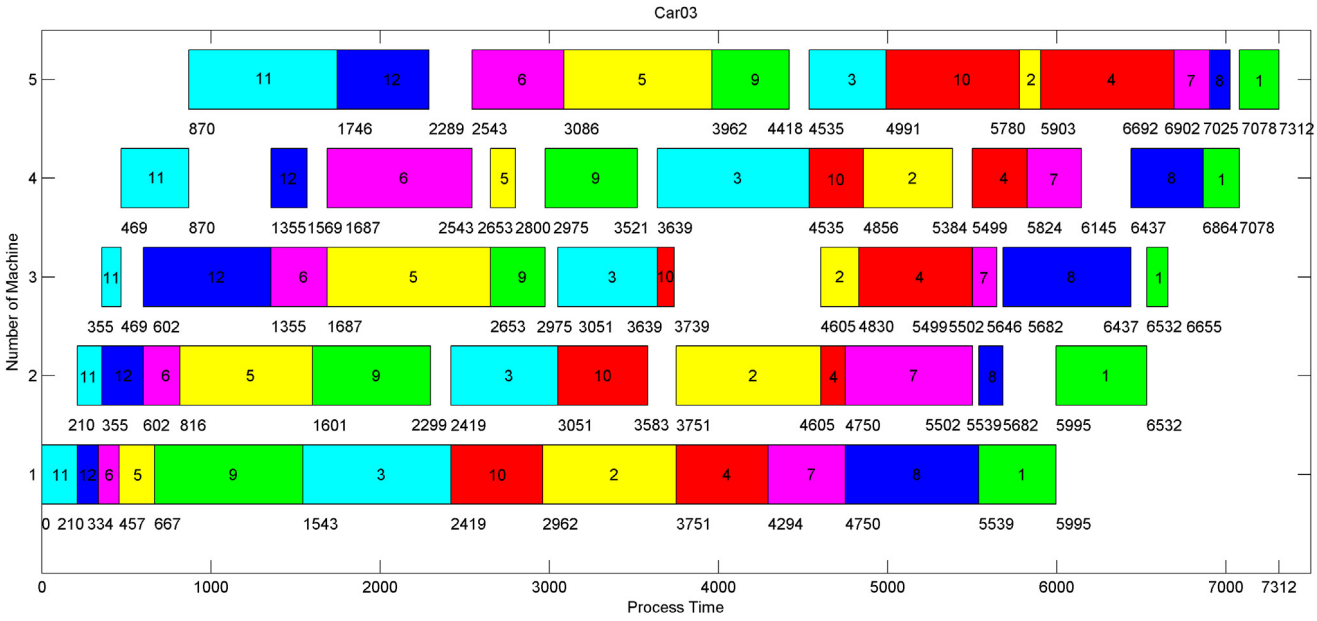


**Fig. 9.** Gantt chart of an optimal schedule for Car03.

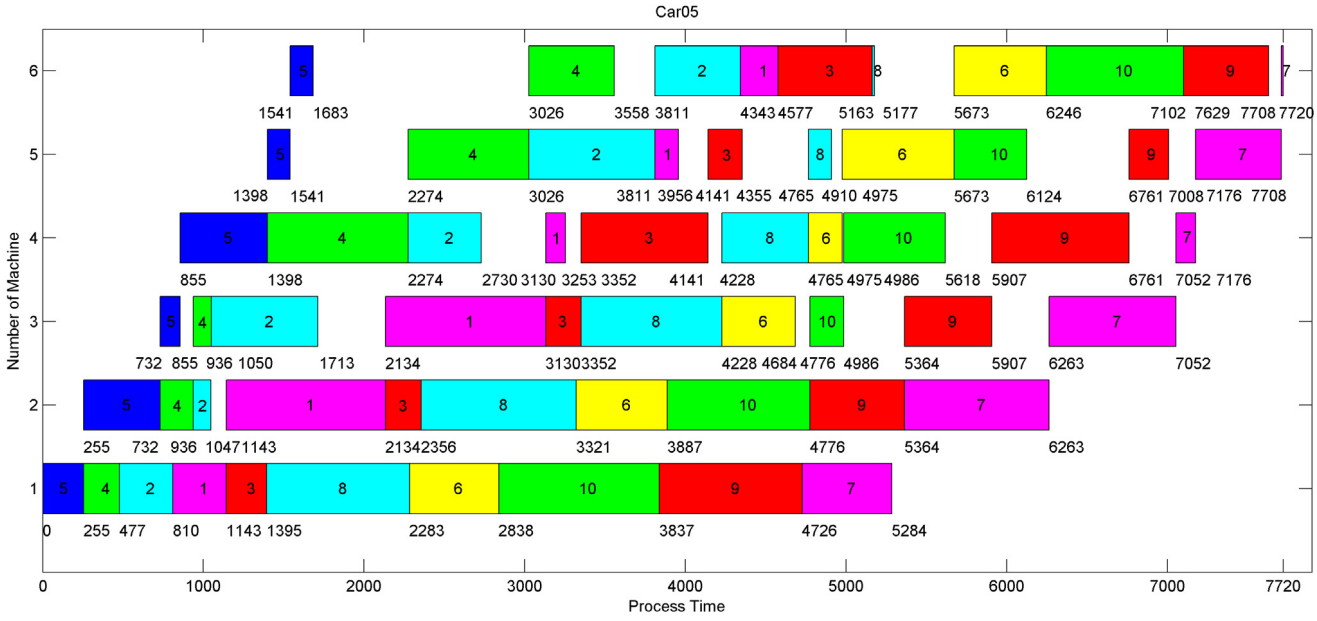


**Fig. 10.** Gantt chart of an optimal schedule for Car05.

To further investigate the performance of the HHS, a simulation is conducted to compare HHS with ATPPSO (Zhang and Sun, 2009) and L-HDE (Liu et al., 2014). Both of them have already achieved better solutions. In Table 17, all the benchmarks are made up of one instance in each combination different from these in Table 16. These benchmarks are including Ta005, Ta010, Ta020, Ta030, Ta050, Ta060, Ta070 and Ta080. All the instances have many peaks and valleys and it is easy to suffer from them. From Table 17, it can be seen that HHS wins the first place in terms of the indicator of ARE, which shows that HHS is more effective than all others. Similarly, Fig. 15 shows the statistical performances resulting from the compared algorithms clearly. From the observation, HHS can obtain smaller makespan than the others. What deserves most is that the results achieved by HHS are far better than those of ATPPSO.

*4.2.6. Comparisons of standard HS, SAOP, ILS, NEGA$_{VNS}$, ACA(best), NACA, FF, NEHI, HLBS and HHS for minimizing makespan on Taillard set*

To comprehensively evaluate the performance of the HHS, HHS is compared with eight state-of-the-art algorithms including SAOP (Osman and Potts, 1989), ILS (Stützle, 1998), NEGA$_{VNS}$ (Zobolas et al., 2009), ACA (Ahmadizar and Barzinpour, 2010), NACA (Ahmadizar, 2012), FF (Fernandez-Viagas and Framinan, 2014), NEHI (Vasiljevic and Danilovic, 2015), HLBS (Chen et al., 2015). The associated results are reported in Table 18. For the sake of convenience and reliability, we do not implement the compared algorithm except for HS and HHS, the rest of results reported in Table 18 come from the corresponding literatures. From Table 18, the result obtained by HHS is significantly better than those of HS, SAOP, ILS, ACA, FF and NEHI except for NEGA$_{VNS}$, NACA and HLBS. For the NACA algorithm, HHS outperforms NACA on half of the instances including 20×5, 20×10, 20×20, 50×5, 100×5 and 100×10.

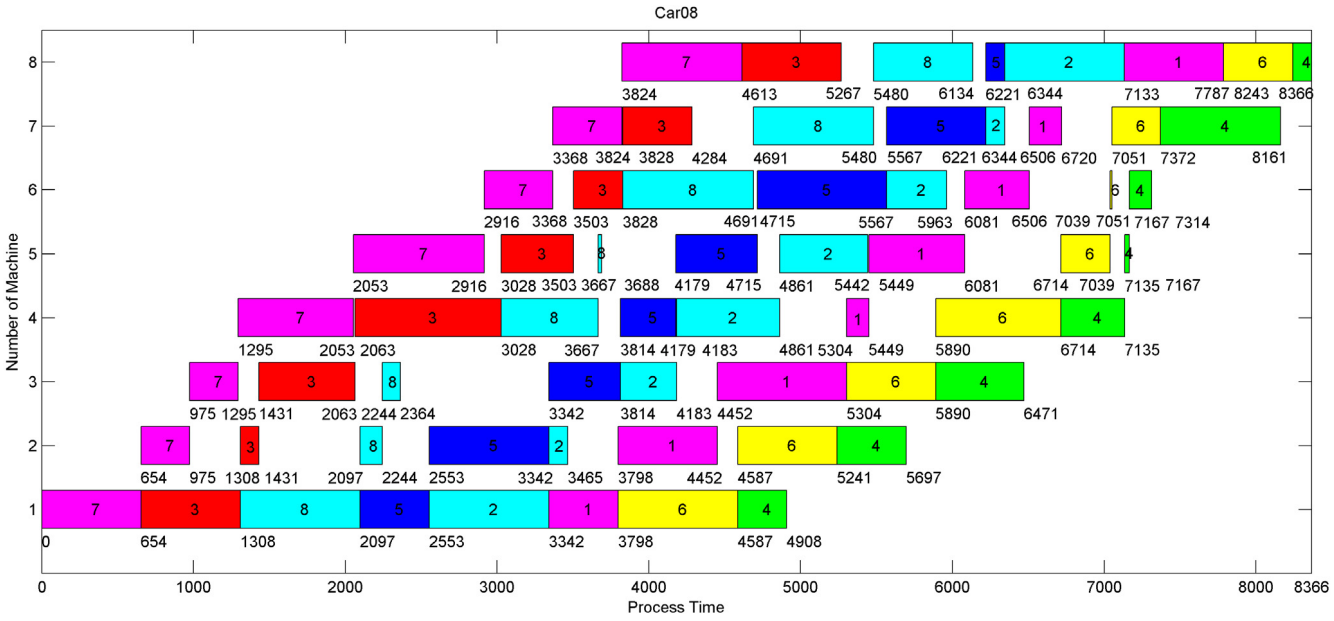


**Fig. 11.** Gantt chart of an optimal schedule for Car08.

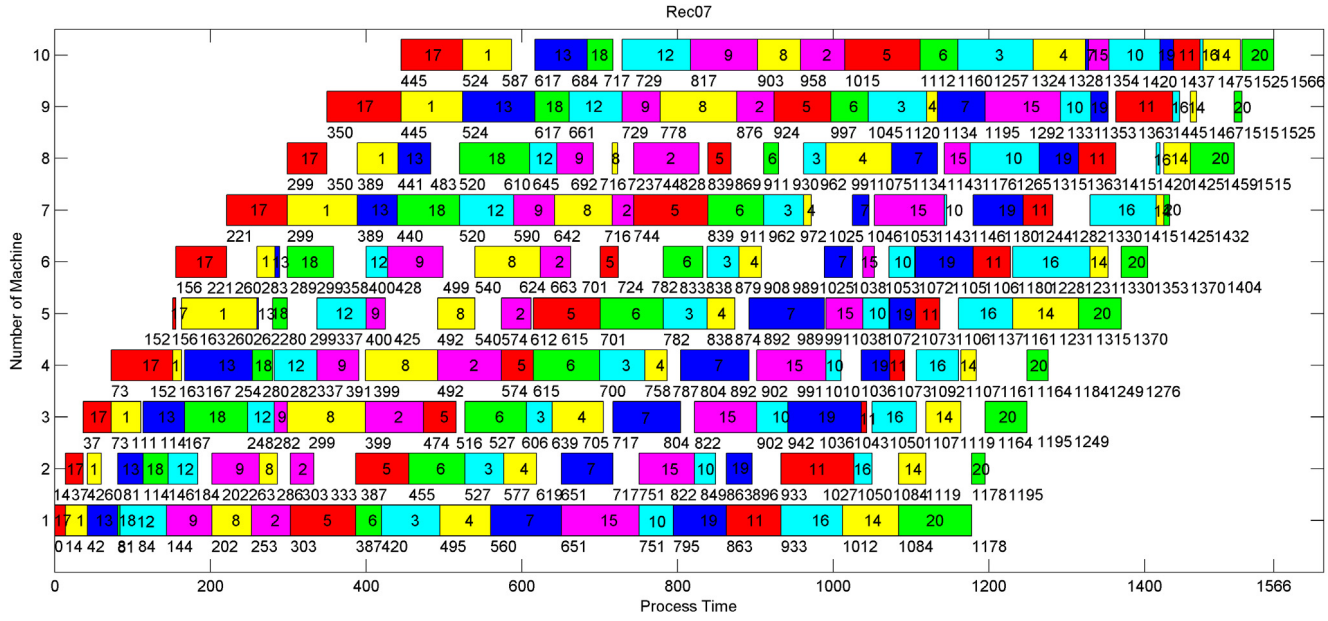


**Fig. 12.** Gantt chart of an optimal schedule for Rec07.

Besides, for the instances $20 \times 20$ and $200 \times 10$, the results achieved by NACA and HHS algorithms have little obvious difference. For the instances $20 \times 10$ and $100 \times 5$, the results achieved by HHS are better than that of $\text{NEGA}_{\text{VNS}}$ and HLBS. What deserves to be mentioned the most is that in this paper HHS not only solves the continuous optimization problems, but also deals with the combinatorial optimization problems. In order to demonstrate the compared algorithms more obviously, a line graph is provided in Fig. 16. The CPU running time of HHS is listed in the last column of Table 15, which can be a reference for other approaches in future research. Although some aforementioned literatures have provided CPU time, we do not compare the CPU running time because the platform, computer language are not same, so the comparison of CPU time makes no sense.

## 5. Conclusion and future work

This paper presents a hybrid harmony search algorithm with efficient job sequence mapping scheme and variable neighborhood search to handle the permutation flow shop scheduling problem with makespan criterion. In the HHS, by investigating the effect of various job sequence mapping scheme, an efficient smallest order value (SOV) rule is introduced to make the HHS suit for addressing PFSSP. Meanwhile, by making use of NEH heuristic and chaotic sequence, an effective initialization scheme is employed to improve the solutions quality of the initial harmony memory. Then an opposition-based learning technique in the selection process and the best individual in HM are adopted to accelerate convergence performance and guide the candidates towards the promising areas. Furthermore, to investigate the properties of HHS, the parameter sensitivity is studied. In addition, based on variable

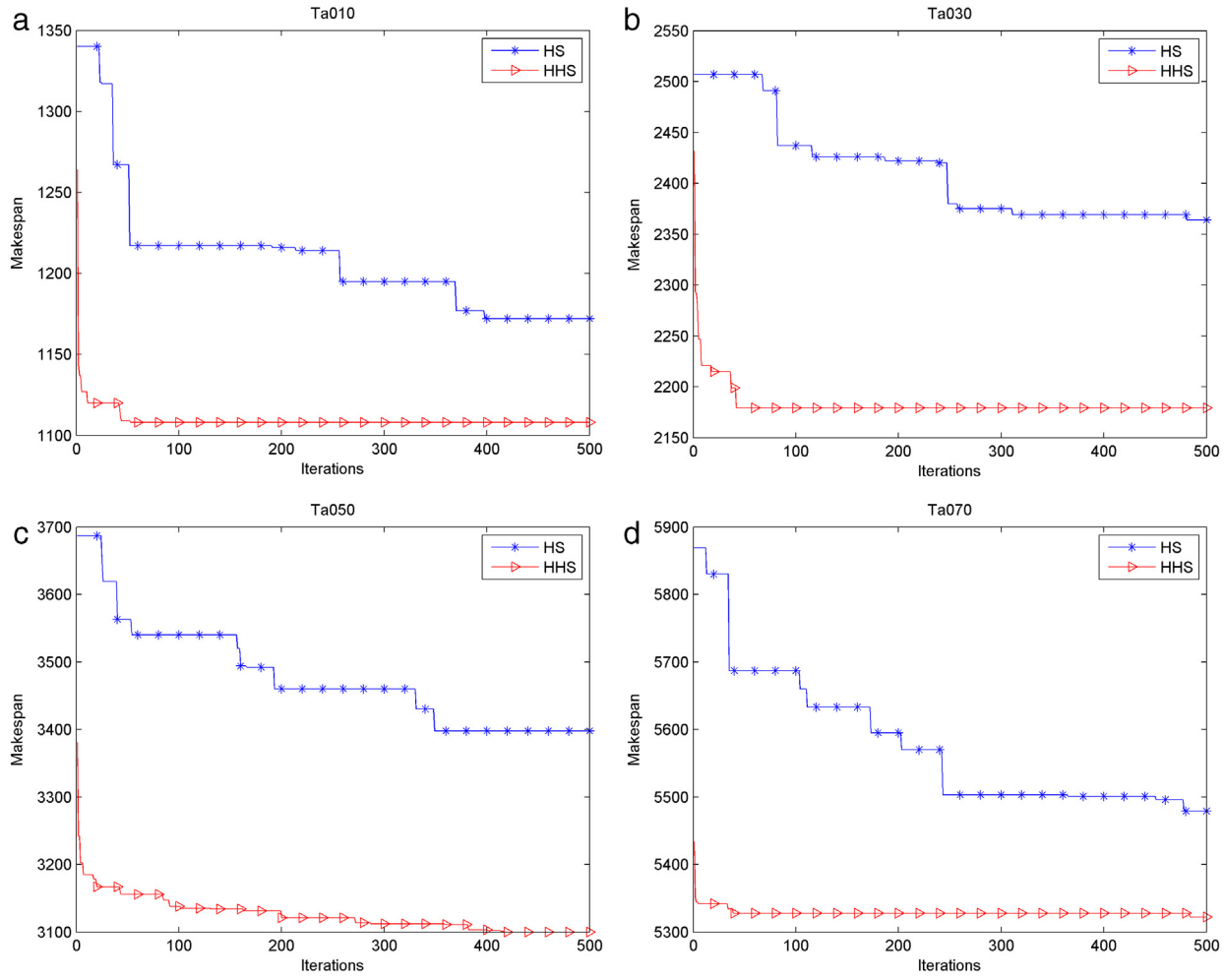


**Fig. 13.** Convergence curves of some typical benchmarks.

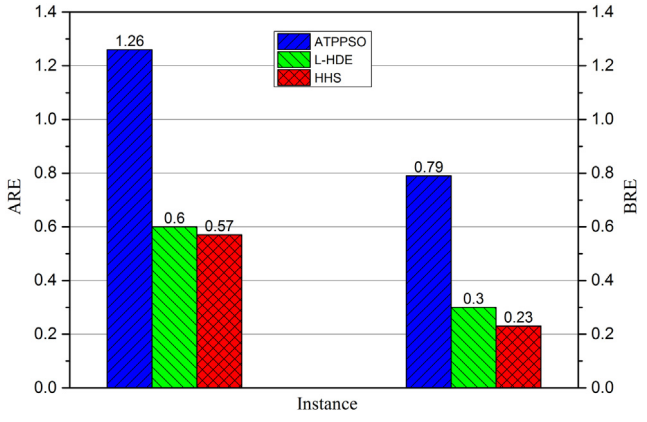


**Fig. 14.** Instance with ARE and BRE.

**Table 19**
Expansions for the abbreviations using in this paper.

| Expansions | Abbreviations |
|---|---|
| The average relative error to $C^*$ | ARE |
| The best relative error to $C^*$ | BRE |
| Bandwidth | BW |
| Flexible job shop scheduling problem | FJSP |
| Flow shop scheduling problems | FSSP |
| Hybrid cuckoo search | HCS |
| Hybrid genetic algorithm | HGA |
| Hybrid harmony search | HHS |
| Harmony memory | HM |
| Harmony memory considering rate | HMCR |
| Harmony memory size | HMS |
| Hybrid variable neighborhood search | HVNS |
| Largest-order-value | LOV |
| Largest-ranked-value | LRV |
| Number of improvisations | NI |
| Pitch adjusting rate | PAR |
| The permutation flow shop scheduling problem | PFSSP |
| Smallest order value | SOV |
| Smallest position value | SPV |
| Variable neighborhood search | VNS |

neighborhood search, the efficient insert and swap structures are embedded into the HHS to adequately enhance local exploitation ability. The statistical results and comparisons based on Carlier's instance set, Reeves and Yamada's instance set as well as Taillard's instance set reveal that HHS is effective and very suitable for solving PFSSP. From Tables 12 and 13, it can be observed that HHS is conspicuously far better than HS, IHS and GHS on all benchmarks. At the same time, to further verify the performance of HHS, HHS is also against the state-of-the-art algorithms including HPSO, ATPPSO, LDPSO and L-HDE. From Tables 14–18, it can be seen that HHS can still work efficiently in most of the cases. To sum up, HHS is an efficient algorithm and it is a good alternative for addressing the permutation flow shop scheduling problem.

However, the superiority of HHS is not very obvious while dealing with large-scale PFSSP. This type of benchmarks has a great number

**Table 20**
Benchmark test functions.

| Benchmark functions | Search space | Optimum |
|---|---|---|
| $\min f1(x) = \sum_{i=1}^{N} xi2$ | $[-100, 100]^n$ | 0 |
| $\min f2(x) = \sum_{i=1}^{N} \lvert xi\rvert + \prod_{i=1}^{N} \lvert xi\rvert$ | $[-10, 10]^n$ | 0 |
| $\min f3(x) = \sum_{i=1}^{N} \left(\sum_{j=1}^{i} xj\right)^2$ | $[-100, 100]^n$ | 0 |
| $\min f4(x) = \sum_{i=1}^{N-1} \left[100\left(xi + 1 - xi^2\right)^2 + (xi - 1)^2\right]$ | $[-30, 30]^n$ | 0 |
| $\min f5(x) = 0.5 + \frac{\sin^2\sqrt{\sum_{i=1}^{D} xi^2} - 0.5}{\left(1+0.001\sum_{i=1}^{D} xi^2\right)^2}$ | $[-100, 100]^n$ | 0 |
| $\min f6(x) = \sum_{i=1}^{N} \left(xi^2 - 10cos\left(2\pi xi\right) + 10\right)$ | $[-5.12, 5.12]^n$ | 0 |
| $\min f7(x) = 418.9829N - \sum_{i=1}^{N} \left(x_i \sin\left(\sqrt{\lvert x_i\rvert}\right)\right)$ | $[-500, 500]^n$ | N = 50, 6.3638e−04 |
| $\min f8(x) = 20 + e - 20exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} xi^2}\right) - \exp\left(\frac{1}{N}\sum_{i=1}^{N} \cos\left(2\pi xi\right)\right)$ | $[-32, 32]^n$ | 0 |
| $\min f9(x) = \frac{1}{4000}\sum_{i=1}^{N} xi^2 - \prod_{i=1}^{N} \cos\left(\frac{xi}{\sqrt{i}}\right) + 1$ | $[-600, 600]^n$ | 0 |
| $\min f10(x) = \sum_{i=1}^{N} x_i^2 + \left(\sum_{i=1}^{N} 0.5ix_i^2\right)^2 + \left(\sum_{i=1}^{N} 0.5ix_i^2\right)^4$ | $[-100, 100]^n$ | 0 |

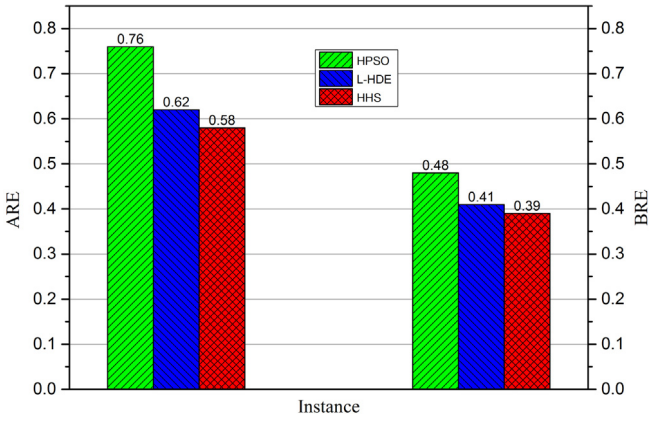


**Fig. 15.** Instance with ARE and BRE.

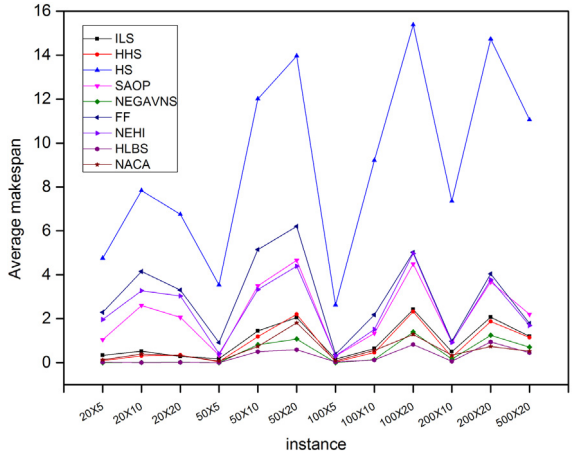


**Fig. 16.** Comparison of different algorithms on TA instance.

of local optima, which are very difficult to achieve to good solutions, especially for Taillard's scheduling problems.

Further work will be conducted in following directions. Firstly, we will consist in improving the performance of HHS and testing its efficiency on large-scale scheduling problems. Secondly, due to large time-consumption in local search, we will try to combine graph structures and neighborhood structures for speeding up the local search. Thirdly, it is desirable to apply the HHS to other varieties of combination optimization problems in the real world.

## Acknowledgments

This work was financially supported by the National Natural Science Foundation of China under grant numbers 61663023. It was also supported by the Key Research Programs of Science and Technology Commission Foundation of Gansu Province (2017GS10817), General and Special Program of the Postdoctoral Science Foundation of China, the Science Foundation for Distinguished Youth Scholars of Lanzhou University of Technology, Lanzhou Science Bureau project under grant numbers 2012M521802, 2013T60889, J201405, and 2013-4-64, respectively.

## Appendix A

See Table 19.

## Appendix B

See Table 20.

## References

Ahmadizar, F., 2012. A new ant colony algorithm for makespan minimization in permutation flow shops. Comput. Ind. Eng. 63, 355–361.
Ahmadizar, F., Barzinpour, F., 2010. A hybrid algorithm to minimize makespan for the permutation flow shop scheduling problem. Int. J. Comput. Intell. Syst. 3, 853–861.
Arul, R., Ravi, G., Velusami, S., 2013. Chaotic self-adaptive differential harmony search algorithm based dynamic economic dispatch. Int. J. Electr. Power Energy Syst. 50, 85–96.
Baskar, A., 2015. A study on minimization of makespan in permutation flow shop scheduling problems.
Beasley, J.E., 1990. OR-Library: distributing test problems by electronic mail. J. Oper. Res. Soc. 1069–1072.
Blazewicz, J., Lenstra, J.K., Kan, A.H.G.R., 1983. Scheduling subject to resource constraints: classification and complexity. Discrete Appl. Math. 5, 11–24.
Carlier, J., 1978. Ordonnancements a contraintes disjonctives. Rev. Fr. Autom. Inform. Rech. Oper. 12, 333–350.
Chen, C.-L., Tzeng, Y.-R., Chen, C.-L., 2015. A new heuristic based on local best solution for permutation flow shop scheduling. Appl. Soft Comput. 29, 75–81.
Del Ser, J., Matinmikko, M., Gil-López, S., Mustonen, M., 2012. Centralized and distributed spectrum channel assignment in cognitive wireless networks: a harmony search approach. Appl. Soft Comput. 12, 921–930.
Della Croce, F., Narayan, V., Tadei, R., 1996. The two-machine total completion time flow shop problem. European J. Oper. Res. 90, 227–237.
Dong, X., Nowak, M., Chen, P., Lin, Y., 2015. Self-adaptive perturbation and multi-neighborhood search for iterated local search on the permutation flow shop problem. Comput. Ind. Eng.
Eberhart, R.C., Kennedy, J., 1995. A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science, New York, NY, pp. 39-43.

Fernandez-Viagas, V., Framinan, J.M., 2014. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. Comput. Oper. Res. 45, 60–67.
Gao, J., Chen, R., Deng, W., 2013. An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. Int. J. Prod. Res. 51, 641–651.
Gao, K.-z., Pan, Q.-k., Li, J.-q., 2011. Discrete harmony search algorithm for the no-wait flow shop scheduling problem with total flow time criterion. Int. J. Adv. Manuf. Technol. 56, 683–692.
Gao, K., Suganthan, P., Pan, Q., Chua, T., Cai, T., Chong, C., 2014. Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives. J. Intell. Manuf. 1–12.
Geem, Z.W., 2007. Optimal scheduling of multiple dam system using harmony search algorithm. In: Computational and Ambient Intelligence. Springer, pp. 316–323.
Geem, Z.W., Kim, J.H., Loganathan, G., 2001. A new heuristic optimization algorithm: harmony search. Simulation 76, 60–68.
Geem, Z.W., Lee, K.S., Park, Y., 2005. Application of harmony search to vehicle routing. Amer. J. Appl. Sci. 2, 1552.
Grabowski, J., Wodecki, M., 2004. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Comput. Oper. Res. 31, 1891–1909.
Hansen, P., Mladenović, N., 2001. Variable neighborhood search: Principles and applications. European J. Oper. Res. 130, 449–467.
Ignall, E., Schrage, L., 1965. Application of the branch and bound technique to some flow-shop scheduling problems. Oper. Res. 13, 400–412.
Ishibuchi, H., Misaki, S., Tanaka, H., 1995. Modified simulated annealing algorithms for the flow shop sequencing problem. European J. Oper. Res. 81, 388–398.
Johnson, S.M., 1954. Optimal two-and three-stage production schedules with setup times included. Nav. Res. Logist. Q. 1, 61–68.
Kuo, I.-H., Horng, S.-J., Kao, T.-W., Lin, T.-L., Lee, C.-L., Terano, T., Pan, Y., 2009. An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model. Expert Syst. Appl. 36, 7027–7032.
Laha, D., Chakraborty, U.K., 2010. Minimising total flow time in permutation flow shop scheduling using a simulated annealing-based approach. Int. J. Autom. Control 4, 359–379.
Li, Y., Li, X., Gupta, J.N., 2015. Solving the multi-objective flowline manufacturing cell scheduling problem by hybrid harmony search. Expert Syst. Appl. 42, 1409–1417.
Li, X., Yin, M., 2013a. A hybrid cuckoo search via Lévy flights for the permutation flow shop scheduling problem. Int. J. Prod. Res. 51, 4732–4754.
Li, X., Yin, M., 2013b. An opposition-based differential evolution algorithm for permutation flow shop scheduling based on diversity measure. Adv. Eng. Softw. 55, 10–31.
Liang, J., Qu, B., Suganthan, P., Hernández-Dıaz, A.G., 2013. Problem Definitions and Evaluation Criteria for the cec 2013 Special Session on Real-Parameter Optimization. Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report 201212.
Liu, H., Gao, L., Pan, Q., 2011. A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem. Expert Syst. Appl. 38, 4348–4360.
Liu, B., Wang, L., Jin, Y.-H., 2007. An effective PSO-based memetic algorithm for flow shop scheduling. IEEE Trans. Syst. Man Cybern. B 37, 18–27.
Liu, Y., Yin, M., Gu, W., 2014. An effective differential evolution algorithm for permutation flow shop scheduling problem. Appl. Math. Comput. 248, 143–159.
Lorenz, E.N., 1963. Deterministic nonperiodic flow. J. Atmos. Sci 20, 25–36.
Mahdavi, M., Fesanghary, M., Damangir, E., 2007. An improved harmony search algorithm for solving optimization problems. Appl. Math. Comput. 188, 1567–1579.
May, R.M., 1976. Simple mathematical models with very complicated dynamics. Nature 261, 459–467.
Mirabi, M., 2014. A novel hybrid genetic algorithm to solve the sequence-dependent permutation flow-shop scheduling problem. Int. J. Adv. Manuf. Technol. 71, 429–437.
Montgomery, D.C., 2008. Design and Analysis of Experiments. John Wiley & Sons.
Moslehi, G., Khorasanian, D., 2014. A hybrid variable neighborhood search algorithm for solving the limited-buffer permutation flow shop scheduling problem with the makespan criterion. Comput. Oper. Res. 52, 260–268.
Nawaz, M., Enscore, E.E., Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11, 91–95.
Nowicki, E., Smutnicki, C., 1996. A fast tabu search algorithm for the permutation flow-shop problem. European J. Oper. Res. 91, 160–175.
Omran, M.G., Mahdavi, M., 2008. Global-best harmony search. Appl. Math. Comput. 198, 643–656.
Osman, I., Potts, C., 1989. Simulated annealing for permutation flow-shop scheduling. Omega 17, 551–557.
Pan, Q.-K., Ruiz, R., 2013. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. Comput. Oper. Res. 40, 117–128.
Pan, Q.-K., Suganthan, P.N., Liang, J.J., Tasgetiren, M.F., 2011a. A local-best harmony search algorithm with dynamic sub-harmony memories for lot-streaming flow shop scheduling problem. Expert Syst. Appl. 38, 3252–3259.
Pan, Q.-K., Wang, L., Gao, L., 2011b. A chaotic harmony search algorithm for the flow shop scheduling problem with limited buffers. Appl. Soft Comput. 11, 5270–5280.
Pinedo, M.L., 2012. Scheduling: Theory, Algorithms, and Systems. Springer Science & Business Media.
Qian, B., Wang, L., Hu, R., Wang, W.-L., Huang, D.-X., Wang, X., 2008. A hybrid differential evolution method for permutation flow-shop scheduling. Int. J. Adv. Manuf. Technol. 38, 757–777.
Rajendran, C., 1993. Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. Int. J. Prod. Econ. 29, 65–73.
Rand, G.K., 1977. Machine scheduling problems: Classification, complexity and computations: A.H.G. RINNOOY KAN, Martinus Nijhoff, The Hague, 1976, ix + 180 pages. European J. Oper. Res. 1, 206.
Reeves, C.R., 1995. A genetic algorithm for flowshop sequencing. Comput. Oper. Res. 22, 5–13.
Reeves, C.R., Yamada, T., 1998. Genetic algorithms, path relinking, and the flowshop sequencing problem. Evol. Comput. 6, 45–60.
Ruiz, R., Maroto, C., 2005. A comprehensive review and evaluation of permutation flowshop heuristics. European J. Oper. Res. 165, 479–494.
Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European J. Oper. Res. 177, 2033–2049.
Stafford, E.F., 1988. On the development of a mixed-integer linear programming model for the flowshop sequencing problem. J. Oper. Res. Soc. 116, 3–1174.
Storn, R., Price, K., 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. 11, 341–359.
Stützle, T., 1998. Applying Iterated Local Search To the Permutation Flow Shop Problem. FG Intellektik, TU Darmstadt, Darmstadt, Germany.
Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Chen, Y.-P., Auger, A., Tiwari, S., 2005. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. KanGAL Report 2005005.
Taillard, E., 1993. Benchmarks for basic scheduling problems. European J. Oper. Res. 64, 278–285.
Tasgetiren, M.F., Liang, Y.-C., Sevkli, M., Gencyilmaz, G., 2007. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European J. Oper. Res. 177, 1930–1947.
Tasgetiren, M.F., Sevkli, M., Liang, Y.-C., Gencyilmaz, G., 2004. Particle swarm optimization algorithm for single machine total weighted tardiness problem. In: Evolutionary Computation, 2004. CEC2004. Congress on. IEEE, p. 1412-1419.
Tavazoei, M.S., Haeri, M., 2007. Comparison of different one-dimensional maps as chaotic search pattern in chaos optimization algorithms. Appl. Math. Comput. 187, 1076–1085.
Valian, E., Tavakoli, S., Mohanna, S., 2014. An intelligent global harmony search approach to continuous optimization problems. Appl. Math. Comput. 232, 670–684.
Vasiljevic, D., Danilovic, M., 2015. Handling ties in heuristics for the permutation flow shop scheduling problem. J. Manuf. Syst. 35, 1–9.
Wang, L., Pan, Q.-K., Tasgetiren, M.Fatih., 2010. Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms. Expert Syst. Appl. 37, 7929–7936.
Wang, S.-y., Wang, L., Liu, M., Xu, Y., 2013. An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. Int. J. Prod. Econ. 145, 387–396.
Xiang, W.-l., An, M.-q., Li, Y.-z., He, R.-c., Zhang, J.-f., 2014. An improved global-best harmony search algorithm for faster optimization. Expert Syst. Appl. 41, 5788–5803.
Xie, Z., Zhang, C., Shao, X., Lin, W., Zhu, H., 2014. An effective hybrid teaching–learning-based optimization algorithm for permutation flow shop scheduling problem. Adv. Eng. Softw. 77, 35–47.
Yuan, Y., Xu, H., Yang, J., 2013. A hybrid harmony search algorithm for the flexible job shop scheduling problem. Appl. Soft Comput. 13, 3259–3272.
Yuan, X., Zhao, J., Yang, Y., Wang, Y., 2014. Hybrid parallel chaos optimization algorithm with harmony search algorithm. Appl. Soft Comput. 17, 12–22.
Zhang, C., Sun, J., 2009. An alternate two phases particle swarm optimization algorithm for flow shop scheduling problem. Expert Syst. Appl. 36, 5162–5167.
Zhao, F., Jiang, X., Zhang, C., Wang, J., 2014a. A chemotaxis-enhanced bacterial foraging algorithm and its application in job shop scheduling problem. Int. J. Comput. Integr. Manuf. 1–16.
Zhao, F., Liu, Y., Zhang, C., Wang, J., 2015a. A Self-adaptive Harmony PSO Search Algorithm and Its Performance Analysis. Expert Syst. Appl.
Zhao, F., Shao, Z., Wang, J., Zhang, C., 2015b. A hybrid differential evolution and estimation of distribution algorithm based on neighbourhood search for job shop scheduling problems. Int. J. Prod. Res. 1–22.
Zhao, F., Tang, J., Wang, J., Jonrinaldi, J., 2014b. An improved particle swarm optimisation with a linearly decreasing disturbance term for flow shop scheduling with limited buffers. Int. J. Comput. Integr. Manuf. 27, 488–499.
Zhao, F., Zhang, J., Wang, J., Zhang, C., 2014c. A shuffled complex evolution algorithm with opposition-based learning for a permutation flow shop scheduling problem. Int. J. Comput. Integr. Manuf. 1–16.
Zhao, F., Zhang, J., Zhang, C., Wang, J., 2015c. An improved shuffled complex evolution algorithm with sequence mapping mechanism for job shop scheduling problems. Expert Syst. Appl. 42, 3953–3966.
Zheng, D.-Z., Wang, L., 2003. An effective hybrid heuristic for flow shop scheduling. Int. J. Adv. Manuf. Technol. 21, 38–44.
Zheng, T., Yamashiro, M., 2010. Solving flow shop scheduling problems by quantum differential evolutionary algorithm. Int. J. Adv. Manuf. Technol. 49, 643–662.
Zobolas, G., Tarantilis, C.D., Ioannou, G., 2009. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. Comput. Oper. Res. 36, 1249–1267.
Zou, D., Gao, L., Wu, J., Li, S., 2010a. Novel global harmony search algorithm for unconstrained problems. Neurocomputing 73, 3308–3318.
Zou, D., Gao, L., Wu, J., Li, S., Li, Y., 2010b. A novel global harmony search algorithm for reliability problems. Comput. Ind. Eng. 58, 307–316.