



A new heuristic based on local best solution for permutation flow shop scheduling



Chun-Lung Chen^{a,*}, Yeu-Ruey Tzeng^b, Chuen-Lung Chen^c

^a Department of Accounting Information, Takming University of Science and Technology, Taipei 114, Taiwan, ROC

^b Department of Computer Science and Information Engineering, Chinese Culture University, Taipei 111, Taiwan, ROC

^c Department of Management Information Systems, National Chengchi University, Taipei 116, Taiwan, ROC

ARTICLE INFO

Article history:

Received 1 October 2012

Received in revised form 18 June 2014

Accepted 6 December 2014

Available online 27 December 2014

Keywords:

Scheduling

Heuristic

Permutation flow shop scheduling

Makespan

ABSTRACT

This paper proposes a population-based heuristic based on the local best solution (HLBS) for the minimization of makespan in permutation flow shop scheduling problems. The proposed heuristic operates through three mechanisms: (i) it introduces a new method to produce a trace-model for guiding the search, (ii) it modifies a filter strategy to filter the solution regions that have been reviewed and guide the search to new solution regions in order to keep the search from trapping into local optima, and (iii) it initiates a new jump strategy to help the search escape if the search is trapped at a local optimum. Computational experiments on the well-known Taillard's benchmark data sets demonstrate that the proposed algorithm generated high quality solutions when compared to existing population-based search algorithms such as genetic algorithms, ant colony optimization, and particle swarm optimization.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

This paper proposes a population-based heuristic based on the local best solution, denoted as HLBS, for the minimization of makespan in a permutation flow shop scheduling problem (PFSP-makespan). The candidate problem determines the best sequence of n jobs that are to be processed on m machines in the same order in order to minimize the completion time of the last job on the last machine (makespan). It has proven to be one of the most studied NP-hard scheduling problems in the strong sense [1] and requires much computational time to find the optimal solution. Therefore, the development of heuristics that find near-optimal solutions in a reasonable computational time has attracted the attention of many researchers in recent decades.

Framinan et al. [2] proposed a classification framework of heuristics for PFSP-makespan. The framework states that the development of a heuristic may consist of three phases: Phase I – index development, Phase II – solution construction and Phase III – solution improvement. A heuristic may consist of one or more of these phases which are generally independent of each other.

In the index development phase, jobs are arranged according to a certain property such as the processing times of each job on each machine. The output of this phase is a ranking of jobs that might be employed either as the input for the next phase or as a solution itself. Dispatching rules such as largest processing time (LPT) and shortest processing time (SPT) are typical of this phase.

In the solution construction phase, a solution is generated in a recursive manner inserting one or more unscheduled jobs in one or more positions of a partial schedule until the solution is completed; therefore, this phase consists of a number of loops.

Palmer [3], CDS [4], Gupta [5], DAN [6], NEH [7] and CDSD [8] are well-known Phase II heuristics. It is worth noting that the NEH heuristic is currently one of the best constructive heuristics. Taillard [9] developed a fast NEH heuristic (NEHT) for PFSP-makespan. Recently, several NEH variants were developed to solve PFSP-makespan [10–13].

In the solution improvement phase, an existing solution is improved by means of some procedures. The two main characteristics of this phase are: (1) an initial solution (input solution) is required and (2) the quality of the final solution is always equal to or better than the quality of the initial solution. Usually, improvement approaches are classified into descending local searches and meta-heuristics. Additionally, Framinan's classification framework shows that heuristics in the solution improvement phase have the longest computation time when compared to the solution construction and index development phases. Phase III heuristics include genetic algorithm (GA) [14–18], ant colony optimization (ACO) [19,20], particle swarm optimization (PSO) [21–25], differential evolution (DE) [26–28], iterated greedy algorithm (IG) [29,30], iterated local search (ILS) [31], simulated annealing (SA) [32–34], tabu search (TS) [35–39] and hybrid metaheuristics [40,41]. Several research papers reviewing heuristics for PFSP-makespan can be found in Framinan et al. [2], Hejazi and Saghafian [42], and Ruiz and Maroto [43].

The proposed HLBS can be regarded as a type of Phase III heuristic. It generates a solution using the heuristic NEHT [9] in the initial iteration and lets the solution be the local best solution. A new trace-model generating rule is applied to the local best solution to generate a trace-model of the local best solution, and a solution construction method is applied to the trace-model to construct a population with a pre-specified number of solutions. A modified filter strategy is applied to the solutions in the population to generate an updated local best solution, and a new jump strategy is applied to help the search escape if the updated local best solution shows that the search traps into a local optimum. The major idea of the proposed algorithm is that the local best solution in an iteration possesses important information about the solution regions searched, so a trace-model generated based on the local best solution should provide valuable information for guiding the search to promising

* Corresponding author. Tel.: +886 2 26585804x5184.

E-mail addresses: charleschen@takming.edu.tw (C.-L. Chen), ZYR3@ulive.pccu.edu.tw (Y.-R. Tzeng), chen6200@gmail.com (C.-L. Chen).

solution regions. In addition, the purpose of the modified filter strategy is to filter the solution regions that have been searched and guide the search to new solution regions. Computational experiments on the well-known Taillard's benchmark data sets [44] will be performed to evaluate the effects of the trace-model, the modified filter strategy, and the jump strategy on the performance of HLBS, and compare the performance of HLBS with other population-based heuristics such as genetic algorithms (GA), ant colony optimization (ACO) and particle swarm optimization (PSO).

The remainder of this paper is organized as follows. Section 2 gives the problem statement, the proposed algorithm HLBS is described in Section 3. Section 4 provides computational experiments, and conclusions and further research of this study are summarized in Section 5.

2. Problem statement

PFSP-makespan can be denoted as $F_m||C_{\max}$ as first described by Graham et al. [45], where F_m represents a flow shop environment of m machines and C_{\max} refers to the makespan. We use the notation proposed by Graham et al. [45]; given a set J of n jobs, a set M of m machines and processing times p_{ij} for each job j on each machine i , the problem consists of scheduling all n jobs at each one of the m machines. All the jobs must be processed in the same processing order and each job j can only start its execution on a machine i if both the previous job on the same machine i and the same job j on the previous machine $i - 1$ have already been processed. The objective of this problem is to determine a job ordering that minimizes the completion time of the last job in the last machine, called the makespan. Although Garey et al. [1] showed that the problem with two-machine can be solved in polynomial time, the general case with m machines is known to be NP-hard. Given a permutation schedule j_1, \dots, j_n for an m -machine flow shop, the completion time of job j_k at machine i , C_{ijk} , can be computed easily through a set of recursive equations:

$$C_{i,j_1} = \sum_{l=1}^i p_{l,j_1}, \quad i = 1, 2, \dots, m \quad (1)$$

$$C_{1,j_k} = \sum_{l=1}^k p_{1,j_l}, \quad k = 1, 2, \dots, n \quad (2)$$

$$C_{i,j_k} = \max(C_{i-1,j_k}, C_{i,j_{k-1}}) + p_{i,j_k}, \quad i = 2, \dots, m; \quad k = 2, \dots, n \quad (3)$$

Then makespan, C_{\max} , is obtained by $C_{\max} = C_{m,j_n}$.

3. The proposed heuristic (HLBS)

Many heuristics based on GA, ACO and PSO have been developed for solving PFSP-makespan. Different population-based heuristics propose different strategies to improve the solutions iteration by iteration. This research proposes a population-based heuristic based on the local best solution, denoted as HLBS, to solve PFSP-makespan. The strategy of HLBS is based on an idea that the local best solution in an iteration possesses important information about the solution regions searched. The basic process of HLBS is presented as follows:

Set $t = 0$. Generate an initial solution using NEHT [9] and let it be the local best solution and the best-so-far solution.

do {

Generate a trace-model based on the local best solution for iteration t .

Generate a new population (M new solutions) by applying a solution construction method to the trace-model.

Apply the modified filter strategy to the M solutions in the new population; update the local best solution.

Launch the jump strategy if the search traps into a local optimum.

$t = t + 1$

} **while** (Not Termination)

Return best-so-far solution.

The HLBS algorithm differs from general population-based heuristics in that it produces only a solution using the heuristic NEHT [9] in the initial iteration and sets the solution to be the local best solution and the best-so-far solution. The major loop in HLBS (do while loop) generates a trace-model based on the local best

Table 1
Example for illustrating the trace-model generating rule.

Position	Job	3	1	2	5	4	Total
		50	100	100	100	100	50
1		50	1	1	1	1	54
2		100	50	1	1	1	153
3		100	100	50	1	1	252
4		100	100	100	50	1	351
5		100	100	100	100	50	450

solution, constructs a new population with M new solutions by applying a solution construction method to the trace-model, and updates the local best solution with the solution produced by applying the modified filter strategy to the M new solutions. If the local best solution is not able to improve the best-so-far solution in a certain number of iterations, it is assumed that the search has trapped into a local optimum. Then the new jump strategy is launched to find a new initial solution as the local best solution and the same loop (do while loop) is performed on the new initial solution. The major components of HLBS: the trace-model generating rule, the solution construction method, the modified filter strategy, and the jump strategy are discussed in detail in the following sections.

3.1. Trace-model generating rule and solution construction method

The new trace-model generating rule is applied to generate a trace-model when the local best solution is updated in every iteration. The following example illustrates the procedure of the generating rule. Given that the updated local best solution in an iteration is $\Pi' = (3, 1, 2, 5, 4)$ and let $\tau(i, u)$ denote the trace-value of job u on position i , the generating rule first assigns a trace-value τ_l to each job on its position in Π' , that is $\tau(1, 3) = \tau(2, 1) = \tau(3, 2) = \tau(4, 5) = \tau(5, 4) = \tau_l$. Then, for each job, the new rule assigns a trace-value τ_p to the positions prior to its position in Π' and assigns a trace-value τ_s to the positions succeeded to its position in Π' . Given that $\tau_p = 1$, $\tau_l = 50$ and $\tau_s = 100$, Table 1 presents the trace-values for all the jobs on different positions. These three trace-values are set to be $\tau_p < \tau_l < \tau_s$ and have to be properly determined to allow the trace-model to keep a job as close as to its position in the local best solution while constructing new solutions so as to keep the valuable information of the sequence of the jobs in every iteration. The following solution construction method will clearly illustrate this idea.

A solution construction in an iteration is composed of job-selections from the first position to the last position for the solution. A revised job-selection rule based on the probabilistic action rule of Dorigo and Gambardella [46] is proposed in HLBS. Given a parameter value q_0 ($0 \leq q_0 \leq 1$), to select a job for position i , the job-selection rule first generates a random number q from a uniform distribution ranged [0,1]; if q is less than or equal to q_0 , then

Eq. (4) is used to select the job; otherwise, a probabilistic action rule (Eq. (5)) is applied to select the job

$$j = \arg \max_{u \in S(i)} \{\tau(i, u)\}, \quad \text{if } q = q_0 \quad (4)$$

$$P(i, j) = \frac{\tau(i, j)}{\sum_{u \in S(i)} \tau(i, u)}, \quad \text{if } q > q_0 \quad (5)$$

where $S(i)$ is the set of unscheduled jobs considered for position i and $P(i, j)$ is the probability for placing job j on position i .

The previous local best solution $\Pi' = (3, 1, 2, 5, 4)$ and the trace-model in Table 1 are used to explain the procedure of the solution construction method. The method constructs a new solution by applying the job-selection rule to the first row of the trace-values in Table 1 to select a job for the first position, to the second row of the trace-values in Table 1 to select a job for the second position, and so on. The procedure for applying the job-selection rule for position 1 is presented as follows. Table 2 summarizes the needed data; it includes the trace-values of the first row in Table 1 and the probabilities for placing the unscheduled jobs on position 1. The probability for placing an unscheduled job on position 1 is calculated by using Eq. (5). Since this is the first position, the unscheduled-job set $S(1) = \{1, 2, 3, 4, 5\}$ and the sum of the trace-values of the jobs in $S(1)$ is $\sum_{u \in S(1)} \tau(1, u) = \tau(1, 1) + \tau(1, 2) + \tau(1, 3) + \tau(1, 4) + \tau(1, 5) = 54$. The probability for placing job 3 on position 1 is $P(1, 3) = \tau(1, 3)/54 = 50/54 = 0.925926$, and for placing the rest of the jobs on position 1 are $P(1, 1) = P(1, 2) = P(1, 4) = P(1, 5) = 1/54 = 0.018519$. Using the data in Table 2, the job-selection rule is implemented: it first randomly generates a number q from a uniform distribution ranged [0,1]; if $q \leq q_0$, since $S(1) = \{1, 2, 3, 4, 5\}$, and $\tau(1, 3) = 50$ and $\tau(1, 1) = \tau(1, 2) = \tau(1, 4) = \tau(1, 5) = 1$, job 3 ($j = \arg \max_{u \in S(1)} \{\tau(1, u)\} = 3$) is selected for position 1; if $q > q_0$, each

job j will be selected with the probability $P(1, j)$ respectively, and the commonly used roulette wheel selection [14] is applied to select a job for position 1. The $P(1, j)$ in Table 2 show that job 3 still has a high probability, $P(1, 3) = 0.925926$, to be selected for position 1. As a result, the probability that job 3 will be retained on position 1 is $q_0 + (1 - q_0) \times P(1, 3)$; if $q_0 = 0.5$, this probability is equal to $0.5 + 0.5 \times 0.925926 = 0.962963$.

Although the probability to keep job 3 on position 1 is high, there still is about a 4% probability that one of the other four jobs may be selected for position 1. If job 5, instead of job 3, is selected for position 1, the job-selection rule is applied to select a job for position 2 using the trace-values of the second row in Table 1. Table 3 presents these trace-values and the probabilities for placing unscheduled jobs ($S(2) = \{1, 2, 3, 4\}$) on position 2. If a randomly generated number $q \leq q_0$, since $\tau(2, 3) = 100$ and $\tau(2, 1) = 50$, job 3 ($j = \arg \max_{u \in S(2)} \{\tau(2, u)\} = 3$) is selected for position 2; this is the

purpose that the trace-values, τ_l and τ_s , are set to be $\tau_l < \tau_s$ in the trace-model. The probability that job 3 will be selected for position 2 is equal to $q_0 + (1 - q_0) \times P(2, 3) = 0.5 + 0.5 \times 0.657895 = 0.828948$. This result shows that if job 3 is not selected for position 1, its position in Π' , it will be selected for position 2 with the highest probability. This illustrates the idea of the trace-model that a job will be placed as close as possible on its position in the local best solution while constructing a new solution.

This simple example reveals that the effects of the trace-model and the solution construction method are highly influenced by the q_0 value and the ratio of τ_p , τ_l and τ_s . A high q_0 value and a large ratio of τ_l and τ_p will cause the job-selection rule to highly retain the job sequence in the local best solution. On the contrary, a low q_0 value and a small ratio of τ_l and τ_p will cause the job-selection rule to drastically change the job sequence in the local best solution and lose the important information embedded in the local best solution. In order to investigate this problem, a study on the relationship among the τ_l , τ_p and τ_s values and a variable q_0 setting method are considered in HLBS. The relationship among the τ_l , τ_p and τ_s values can be described using two simple equations: $\tau_l = \tau_p \times x$ and $\tau_s = \tau_l + \tau_p \times y = \tau_p \times x + \tau_p \times y = \tau_p \times (x + y)$, so

the relationship among the τ_l , τ_p and τ_s values can be determined by the two parameters x and y . As the values of these two parameters x and y become larger, there exists a higher possibility that the job sequence in the local best solution will be retained in constructing new solutions. Therefore, a number of the combinations of x and y will be considered in order to study the effect of the trace-model on the performance of HLBS.

In addition, a block property of PFSP-makespan is applied in the construction method. Several recent works [39,47,48] have shown that the block property of the PFSP-makespan can be developed and used to reduce the size of neighborhood. Therefore, the block property of PFSP-makespan will be considered in the construction method to improve the efficiency of HLBS. A solution of a PFSP-makespan problem can be presented as a PERT graph, and the length of the critical path of the graph is the makespan of the solution. A block is a sequence of consecutive jobs on a machine in a critical path; therefore, if a PFSP-makespan has m machines, the critical path of a solution will have m blocks. To apply the block property in the construction method for a PFSP-makespan problem with m machines, the HLBS will construct m solutions in each iteration. The first solution is constructed by choosing the first block from the local best solution and applying equations (4) and (5) to determine the jobs for the rest of the positions in the solution; the second solution is constructed by choosing the second block from the local best solution and applying equations (4) and (5) to determine the jobs for the rest of the positions in the solution and so forth. Since the number of positions to be filled out while constructing a solution is decreased, applying the block property in the construction method will improve the efficiency of the HLBS.

3.2. Modified filter strategy

Local search methods are crucial for improving the effectiveness of population-based heuristics. They usually are applied to the best solution in an iteration or the global best solution to improve the quality of the solution; however, this may cause a search trap into the local optima. Tzeng et al. [49] proposed a filter strategy to address this problem. The filter strategy is applied when all the members (M) finish constructing their solutions in an iteration. Its purpose is to filter the solution regions that have been reviewed and guide the search to new solution regions in order to keep the search from trapping into local optima. A filter-list, defined as a first-in, first-out queue, is used to store the makespan of the chosen solution in each iteration and a parameter, f -size, is defined as the size of the queue. The queue is set to be empty initially. When all the M solutions are constructed, the solutions are sorted according to their makespans in ascending order, and the filter strategy is applied from the top of the M solutions until the first solution, whose makespan is different from all the makespans in the filter-list, is found and stored the makespan of the solution in the filter list. If none of the M solutions has a different makespan from the makespans in the filter-list, the last of the M solutions is chosen (but the makespan will not be stored in the filter-list). The purpose of comparing makespans instead of job-sequences of solutions while using the filter strategy is two-fold. Firstly, it may guide the search to the solution regions which have not been examined. Secondly, it can significantly reduce computation time by comparing the solution constructed by a member and the solutions stored in the filter-list; this is especially critical when the number of jobs considered in a problem is large. In addition, the idea of choosing the solution with the largest makespan when none of the M solutions has a different makespan from the makespans in the filter-list is that it may prevent the search of HLBS from quick convergence.

Once a solution is chosen using the filter strategy, the local search method (denoted as NEHT_LS) is applied to improve the makespan of the solution. NEHT_LS integrates Taillard's

Table 2

Trace-values and probabilities for applying the job-selection rule for position 1.

Unscheduled jobs ($S(1)$)	3	1	2	5	4	Total
Position 1	50	1	1	1	1	54
$P(1,j)$	0.925926	0.018519	0.018519	0.018519	0.018519	1.00

Table 3

Trace-values and probabilities for applying the job-selection rule for position 2.

Unscheduled jobs ($S(2)$)	3	1	2	4	Total
Position 2	100	50	1	1	152
$P(2,j)$	0.657895	0.328947	0.006579	0.006579	1.00

Modified-NEH method [9] with Ruiz and Stützle's [29] iterative improvement method. Given that Π is the job sequence of the chosen solution, NEHT_LS first randomly chooses a job k and removes it from Π ; then it inserts job k into the first position, the last position, and the positions between every two consecutive jobs in Π to generate n different solutions, and lets Π'' be the best of the n generated solutions. If the makespan of Π'' is smaller than that of Π , NEHT_LS will update Π with Π'' and will repeat the same procedure until Π cannot be further improved. If the makespan of Π is smaller than that of the local best solution, it will update the local solution with Π ; if the makespan of Π is smaller than that of the best-so-far solution, it will update the best-so-far solution with Π . The procedure integrating the filter strategy and NEHT_LS is denoted as filtered local search (FLS) in this research.

The modified filter strategy first implements FLS, then determines if the makespan of the schedule generated by FLS dominates the best-so-far solution. If so, it will stop; otherwise, it will implement FLS one more time by using the filter strategy to find a solution different from the one found by the filter strategy in the first FLS.

3.3. Jump strategy

The main idea of the jump strategy is to guide the search to jump to another solution region when the search is trapped in a local optimum. We define the search trapped in a local optimum when the search is not able to improve the best-so-far solution in a number of iterations. The solution generated by the jump strategy is considered to be a new initial solution, and the search procedure is restarted.

The jump strategy proposed in this paper first applies the Destruction and Construction Operation [29] to the detected local optimum M times to generate M new solutions. The solution with the minimum makespan, which satisfies the following conditions: (i) the makespan is less than or equal to a pre-determined objective-value distance and (ii) the job sequence of the solution is different from the job sequence of the local optimum, is chosen and used as the new initial solution. The objective-value distance is defined to be the distance of a jump from the objective value of the current local best solution and is calculated by multiplying the objective value of the current local best solution with a parameter, *jump-rate*. If none of the M solutions satisfies the conditions, the same procedure will be implemented until a solution is produced. In order to apply the Destruction and Construction Operation to a schedule, S , first randomly choose n_1 jobs from S and let the job sequence of the n_1 jobs be s_1 and the job sequence of the rest of the jobs in S be s_2 . Then, insert the first job in s_1 into the first position, the last position and the positions between every two consecutive jobs in s_2 and choose the sequence with the smallest makespan; repeat the same process until all the n_1 jobs in s_1 are inserted in s_2 . In this paper, the Destruction and Construction Operation is implemented three times with $n_1 = 5$.

4. Computational experiments of HLBS

The well-known Taillard's test problems for PFSP-makespan [44] are used to evaluate the performance of HLBS. The test problems are composed of 12 different problem sets with different numbers of jobs (n) and different numbers of machines (m) and 10 instances are included in each problem set. Due to computational burden, twelve instances, selecting the first instance from each of the 12 problem sets, denoted as *Test1*, are used to investigate the effects of the major components of HLBS: the trace-model generating rule, the solution construction method, the modified filter strategy, and the jump strategy. Note that the parameters defined for these components are: q_0 , x , y , *f-size* and *jump-rate*, respectively. Therefore, experiments will be designed based on the parameters to investigate the effects of the components. Then, HLBS with the best combination of the parameters will be applied to solve all the test problems, and its performance will be compared with promising population-based heuristics such as genetic algorithms (GA), ant colony optimization (ACO) and particle swarm optimization (PSO). All the algorithms in this paper are coded in C language and executed on the Linux operating system.

Table 4 summarizes the three levels considered for each of the five parameters of HLBS: 0.5, 0.7, 0.9 for q_0 ; 1, 50 and 100 for x ; 200, 400 and 600 for y ; none, 7 and 14 for *f-size*; none, 0.02 and 0.04 for *jump-rate*. Note that none for *f-size* refers to no modified filter strategy is applied and none for *jump-rate* refers to no jump strategy is applied. The remaining parameters of HLBS are the number of iterations without improvement for defining trapping at a local optimum and the termination criterion. The first parameter is determined by trial-and-error and set to be the number of machines of the instances solved, and the execution time, like most of the other researches, is chosen to be the termination criterion. Therefore, there are a total of 243 different combinations of the five parameters.

The HLBS is applied with each of the 243 combinations to solve the 12 instances in *Test1* with limited computation times, $n \times (m/2) \times 30$ milliseconds [16], for thirty trials. The average relative performance (ARP) is used to measure the performance of the HLBS. The formula of ARP is as follows: $ARP = \sum_{i=1}^R \left(\frac{solution_i - Best_{sol}}{Best_{sol}} \times 100 \right) / R$; given an instance, $solution_i$ is the makespan obtained by trial i of the HLBS with a combination of the

Table 4

Experimental parameters.

Parameters	Levels	Total levels
x	1, 50 and 100 200, 400 and 600	3
y	200, 400 and 600	3
q_0	0.5, 0.7 and 0.9	3
<i>f-size</i>	None, 7, 14	3
<i>Jump-rate</i>	None, 0.02, 0.04	3

Table 5

ANOVA table for testing the significance of the five parameters.

Source	Type III sum of squares	df	Mean square	F	Sig.
q_0	0.003	2	0.002	0.205	0.815
x	0.136	2	0.068	8.763	0.000
y	0.036	2	0.018	2.343	0.096
$f\text{-size}$	0.318	2	0.159	20.445	0.000
$Jump\text{-rate}$	2.204	2	1.102	141.628	0.000
Instance	1160.122	11	105.466	13,552.321	0.000
Error	22.521	2894	0.008		
Corrected total	1185.342	2915			

Note: Adjusted R Squared = 1 – (Error/Corrected total) = 1 – (22.521/1185.342) = 0.981.

Table 6

Results of Duncan's test for the significant parameters.

x	Average ARP	Subset	
		1	2
50	0.542038	A	
	0.542370	A	
	0.556709	B	
y	Average ARP	Subset	
		1	2
600	0.542915	A	
	0.546650	A	B
	0.551551	B	
f-size	Average ARP	Subset	
		1	2
7	0.538866	A	
	0.540468	A	
	0.561783	B	
Jump-rate	Average ARP	Subset	
		1	2
0.04	0.526964	A	
	0.528,238	A	
	0.6280	B	

parameters for the instance, and $Best_{sol}$ is the best makespan that all the research has found for the instance provided by Zobolas et al. [40]. The computation time for applying HLBS with a combination of the parameters to solve the instance, 500×20 , with 30 trials is 4500 s and is 9877.5 s for solving all the 12 instances; therefore, the computation time for executing HLBS with all the 243 combinations of the parameters is approximately 28 days.

The analysis of variance (ANOVA) is applied to analyze the ARPs produced. Table 5 presents the results of the ANOVA table. The results show that all the parameters, except q_0 , significantly affect the ARP of the test problems. Therefore, the Duncan's test is applied to test all the significant parameters. Table 6 summarizes the results of the Duncan tests: the minimum average ARP for each parameter is: $x=50$, $y=600$, $f\text{-size}=7$ and $jump\text{-rate}=0.04$. This condition is very close to the condition that generates the best solution: $q_0=0.9$, $x=50$ and $y=600$, $f\text{-size}=7$ and $jump\text{-rate}=0.04$. Since the difference between the average ARP of $q_0=0.7$ (0.546257) and the average ARP of $q_0=0.9$ (0.546344) is negligible, the best combination of the five parameters for HLBS is determined to be $q_0=0.9$, $x=50$, $y=600$, $f\text{-size}=7$ and $jump\text{-rate}=0.04$.

A further experiment is performed to evaluate the effect of the trace-model generating rule and the solution construction method on HLBS. We replace the trace-model generating rule and the solution construction method with the commonly used

Table 7

Computational results of H-SWAP, H-INSERT and HLBS.

Test problems	H-SWAP	H-INSERT	HLBS
20 × 5	0.04	0.03	0.03
20 × 10	0.00	0.00	0.00
20 × 20	0.02	0.03	0.02
50 × 5	0.00	0.00	0.00
50 × 10	0.63	0.6	0.61
50 × 20	0.84	0.87	0.83
100 × 5	0.04	0.04	0.04
100 × 10	0.29	0.26	0.21
100 × 20	1.29	1.29	1.21
200 × 10	0.17	0.16	0.08
200 × 20	1.4	1.4	1.29
500 × 20	0.69	0.71	0.60
Average	0.45	0.45	0.41

Table 8

Results of Wilcoxon signed ranks test for H-SWAP, H-INSERT and HLBS.

Algorithm	Test statistics	
	Z	Asymp. Sig. (2-tailed)
H-SWAP vs. HLBS	2.530	0.011
H-INSERT vs. HLBS	2.319	0.02

SWAP or INSERT operation in HLBS. Given a job-sequence, a SWAP operation randomly chooses two jobs from the job-sequence and interchanges their positions, and an INSERT operation randomly chooses a job from the job-sequence and randomly inserts the job into the first position, the last position, or a position between every two consecutive jobs in the job-sequence. The HLBS using SWAP operation is denoted as H_SWAP and The HLBS using INSERT operation is denoted as H_INSERT. The HLBS, H_SWAP and H_INSERT are applied to solve all the 120 instances in the 12 problem sets with limited computation times, $n \times (m/2) \times 30$ ms, for ten trials, and a nonparametric test, Wilcoxon Signed Ranks, is applied to compare the performance of HLBS, H_SWAP and H_INSERT. Table 7 presents the average ARPs produced by HLBS, H_SWAP and H_INSERT for the twelve problem sets and Table 8 presents the results of all the Wilcoxon Signed Ranks tests. The results show that HLBS significantly dominates H_SWAP at a significance level of 0.011 and significantly dominates H_INSERT at a significance level of 0.02. This is especially true for larger size problems such as 100×20 , 200×20 and 500×20 .

HLBS with the best combination of the parameters is then applied to solve all the 120 test problems, and its performance is compared with a ACO algorithms, PACO [19], a PSO algorithm, PSOvns and two hybrid GA related heuristics, NEGAvns [40] and HGA.RMA [16], which reported promising solutions for PFSP-makespan. Ruiz et al. [16] compared the performance of PACO and HGA.RMA based on the same number of replication runs ($R=5$) and the same computation times: $n \times (m/2) \times 30$, $n \times (m/2) \times 60$, and $n \times (m/2) \times 90$ ms. All the algorithms were run on a PC with Intel Pentium IV at 2.8 GHz. Therefore, we compare the performance of HLBS with PACO and HGA.RMA based on the same computation times using a PC with the same computing power. Tables 9–11 present the average ARPs produced by PACO, HGA.RMA and HLBS for the twelve problem sets with each of the three computation times respectively.

The Wilcoxon Signed Ranks test is applied to test if the performance of HLBS significantly dominates PACO and HGA.RMA respectively. Table 12 summarizes the results of all the Wilcoxon Signed Ranks tests. The results show that HLBS significantly dominates PACO and HGA.RMA under all the different computation times.

Table 9Computational results of PACO, HGA_RMA and HLBS ($t30^a$).

Test problems	PACO	HGA_RMA	HLBS
20 × 5	0.20	0.05	0.04
20 × 10	0.32	0.10	0.00
20 × 20	0.31	0.10	0.04
50 × 5	0.08	0.00	0.00
50 × 10	0.90	0.77	0.65
50 × 20	1.46	1.19	0.93
100 × 5	0.04	0.02	0.04
100 × 10	0.35	0.26	0.22
100 × 20	2.17	1.59	1.32
200 × 10	0.26	0.16	0.11
200 × 20	2.00	1.42	1.41
500 × 20	0.98	0.87	0.63
Average	0.756	0.55	0.45

^a $t30 = n \times (m/2) \times 30$ ms.**Table 10**Computational results of PACO, HGA_RMA and HLBS ($t60^a$).

Test problems	PACO	HGA_RMA	HLBS
20 × 5	0.16	0.03	0.03
20 × 10	0.30	0.09	0.00
20 × 20	0.15	0.07	0.01
50 × 5	0.03	0.01	0.00
50 × 10	0.87	0.64	0.55
50 × 20	1.39	1.07	0.81
100 × 5	0.03	0.01	0.04
100 × 10	0.32	0.23	0.18
100 × 20	1.99	1.33	1.12
200 × 10	0.26	0.13	0.08
200 × 20	1.86	1.30	1.18
500 × 20	0.92	0.76	0.55
Average	0.690	0.47	0.38

^a $t60 = n \times (m/2) \times 60$ ms.**Table 11**Computational results of PACO, HGA_RMA and HLBS ($t90^a$).

Test problems	PACO	HGA_RMA	HLBS
20 × 5	0.18	0.04	0.03
20 × 10	0.24	0.02	0.00
20 × 20	0.18	0.05	0.01
50 × 5	0.05	0.00	0.00
50 × 10	0.81	0.72	0.54
50 × 20	1.41	0.99	0.74
100 × 5	0.02	0.01	0.04
100 × 10	0.29	0.16	0.15
100 × 20	1.93	1.30	1.00
200 × 10	0.23	0.14	0.08
200 × 20	1.82	1.26	1.07
500 × 20	0.85	0.69	0.49
Average	0.668	0.45	0.35

^a $t90 = n \times (m/2) \times 90$ ms.**Table 12**

Results of the Wilcoxon signed ranks test for PACO, HGA_RMA and HLBS under different computation times.

Time	Algorithm	Test statistics	
		Z	Asymp. Sig. (2-tailed)
t30	PACO vs. HLBS	2.934	0.003
	HGA_RMA vs. HLBS	2.669	0.008
t60	PACO vs. HLBS	2.982	0.003
	HGA_RMA vs. HLBS	2.711	0.007
t90	PACO vs. HLBS	2.982	0.003
	HGA_RMA vs. HLBS	2.581	0.010

Table 13Computational results of HGA_RMA, NEGAvns, PSOvns and HLBS ($t = n \times m/10$ s).

Test problems	NEGAvns	PSOvns	HLBS
20 × 5	0.00	0.03	0.02
20 × 10	0.01	0.02	0.00
20 × 20	0.02	0.05	0.01
50 × 5	0.00	0.00	0.00
50 × 10	0.82	0.57	0.50
50 × 20	1.08	1.36	0.59
100 × 5	0.00	0.00	0.03
100 × 10	0.14	0.18	0.12
100 × 20	1.40	1.45	0.83
200 × 10	0.16	0.18	0.06
200 × 20	1.25	1.35	0.95
500 × 20	0.71	^a	0.45
Average	0.466	0.472	0.300

^a The authors do not provide results for the 500 × 20 instance group.**Table 14**Results of Wilcoxon signed ranks test for NEGAvns, PSOvns and HLBS ($t = n \times m/10$ s).

Algorithm	Test statistics	
	Z	Asymp. Sig. (2-tailed)
NEGAvns vs. HLBS	2.180	0.029
PSOvns vs. HLBS	2.497	0.013

Table 13 presents the average ARPs generated by NEGAvns, PSOvns and HLBS on a PC with Intel Pentium IV at 2.4 GHz under the same computation time, $n \times m/10$ s, and the same number of replication runs ($R = 10$) [40]. The Wilcoxon Signed Ranks test is also applied to compare the performance between HLBS and each of the algorithms: NEGAvns and PSOvns. **Table 14** summarizes the results of all the Wilcoxon Signed Ranks tests. The results show that HLBS significantly dominates NEGAvns and PSOvns.

5. Conclusions and further research

This paper proposes a population-based heuristics based on the local best solution, HLBS, for the permutation flow shop scheduling problem (PFSP-makespan). The computational results have shown that HLBS is an effective heuristic for PFSP-makespan. It dominated all the promising population-based heuristics related to ACO, PSO and GA (PACO, PSOvns, HGA_RMA, NEGAvns). The results also showed that the proposed trace-model, solution construction method, modified filter strategy and jump strategy significantly influence the performance of HLBS. Furthermore, since the flow shop problem is a special case of the flexible flow line problem and the job shop problem, the proposed heuristic can also be applied towards these two problems. Additionally, the proposed algorithm can be applied to single machine problems.

In addition, a few important observations made in this research should be noted. First, although HLBS with $q_0 = 0.9$, $x = 50$, $y = 600$ performed well for solving all the test problems, its performance was not stable for solving hard test problems (50 × 20, 100 × 20 and 200 × 20) [50]. Computational results show that the parameter set for producing the best average ARP for each of the hard test problems differs from each other. Future studies focused on optimizing these parameters using metaheuristics such as genetic algorithms [51] are warranted. Second, although the jump strategy worked effectively for HLBS, it could not guarantee that a jump could jump out of the region of a local optimum. This would cause a search to converge to the same local optimum after the jump strategy was applied. This apparent limitation should be further investigated in order to develop novel mechanisms that can guarantee escape from the local optimum thus improving the effectiveness and efficiency of HLBS.

Acknowledgements

This paper was supported in part by the National Science Council, Taiwan, ROC, under the contract NSC100-2221-E-004 -004. The authors are grateful to the anonymous referees for their constructive comments that have greatly improved the presentation of this paper.

References

- [1] M.R. Garey, D.S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, *Math. Oper. Res.* 1 (1976) 117–129.
- [2] J. Framan, K.N.D. Gupta, R. Leisten, A review and classification of heuristics for the permutation flowshop with makespan objective, *J. Oper. Res. Soc.* 55 (2004) 1243–1255.
- [3] D.S. Palmer, Sequencing jobs through a multi-stage process in the minimum total time – a quick method of obtaining a near optimum, *Oper. Res. Q.* 16 (1965) 101–107.
- [4] H.G. Campbell, R.A. Dudek, M.L. Smith, A heuristic algorithm for the n -job, m -machine sequencing problem, *Manag. Sci.* 16 (1970) 630–637.
- [5] J.N.D. Gupta, A functional heuristic algorithm for the flow-shop scheduling problem, *Oper. Res. Q.* 22 (1971) 39–47.
- [6] D.G. Dannenbring, An evaluation of flow shop sequencing heuristics, *Manag. Sci.* 23 (1977) 1174–1182.
- [7] M. Nawaz, E.E. Enscore, I. Ham, A heuristic algorithm for the m -machine, n -job flow-shop sequencing problem, *Omega* 11 (1983) 91–95.
- [8] Y.B. Park, C.D. Pegden, E.E. Enscore, A survey and evaluation of static flow shop scheduling heuristics, *Int. J. Prod. Res.* 22 (1984) 127–141.
- [9] E. Taillard, Some efficient heuristic methods for the flow shop sequencing problem, *Eur. J. Oper. Res.* 47 (1990) 65–74.
- [10] X.Y. Dong, H.K. Huang, P. Chen, An improved NEH-based heuristic for the permutation flowshop problem, *Comput. Oper. Res.* 35 (2008) 3962–3968.
- [11] P.J. Kalczynski, J. Kamburowski, An improved NEH heuristic to minimize makespan in permutation flow shops, *Comput. Oper. Res.* 35 (2008) 3001–3008.
- [12] S. Farahmand Rad, R. Ruiz, N. Borojerdi, New high performing heuristics for minimizing makespan in permutation flowshops, *Omega* 37 (2009) 331–345.
- [13] V. Fernandez-Viagas, J.M. Framan, On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem, *Comput. Oper. Res.* 45 (2014) 60–67.
- [14] C.L. Chen, V.S. Vempati, N. Aljaber, An application of genetic algorithms for flow shop problems, *Eur. J. Oper. Res.* 80 (1995) 389–396.
- [15] C.R. Reeves, T. Yamada, Genetic algorithms, path relinking and the flowshop sequencing problem, *Evol. Comput.* 6 (1998) 45–60.
- [16] R. Ruiz, C. Maroto, J. Alcaraz, Two new robust genetic algorithms for the flowshop scheduling problem, *Omega* 34 (2006) 461–547.
- [17] Y.M. Chen, M.C. Chen, P.C. Chang, S.H. Chen, Extended artificial chromosomes genetic algorithm for permutation flowshop scheduling problems, *Comput. Ind. Eng.* 62 (2012) 536–545.
- [18] S.H. Chen, P.C. Chang, T.C.E. Cheng, Q. Zhang, A self-guided genetic algorithm for permutation flowshop scheduling problems, *Comput. Oper. Res.* 39 (2012) 1450–1457.
- [19] C. Rajendran, H. Ziegler, Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs, *Eur. J. Oper. Res.* 155 (2004) 426–438.
- [20] K.C. Ying, C.J. Liao, An ant colony system for permutation flow-shop sequencing, *Comput. Oper. Res.* 31 (2004) 791–801.
- [21] Z. Lian, X. Gu, B. Jiao, A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan, *Appl. Math. Comput.* 175 (2006) 773–785.
- [22] C.J. Liao, C.T. Tseng, P. Luarn, A discrete version of particle swarm optimization for flowshop scheduling problems, *Comput. Oper. Res.* 34 (2007) 3099–3111.
- [23] I.H. Kuo, S.J. Horng, T.W. Kao, T.L. Lin, C.L. Lee, T. Terano, Y. Pan, An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model, *Expert Syst. Appl.* 36 (2009) 7027–7032.
- [24] J. Zhang, C. Zhang, S. Liang, The circular discrete particle swarm optimization algorithm for flow shop scheduling problem, *Expert Syst. Appl.* 37 (2010) 5827–5834.
- [25] X. Wang, L. Tang, A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking, *Appl. Soft Comput.* 12 (2012) 652–662.
- [26] N. Andreas, S. Omirou, Differential evolution for sequencing and scheduling optimization, *J. Heurist.* 12 (2006) 395–411.
- [27] G. Onwubolu, D. Davendra, Scheduling flow shops using differential evolution algorithm, *Eur. J. Oper. Res.* 171 (2006) 674–692.
- [28] Q.K. Pan, M.F. Tasgetiren, Y.C. Liang, A discrete differential evolution algorithm for the permutation flowshop scheduling problem, *Comput. Ind. Eng.* 55 (2008) 795–816.
- [29] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, *Eur. J. Oper. Res.* 177 (2007) 2033–2049.
- [30] D. Khorasanian, G. Mosleh, An iterated greedy algorithm for solving the blocking flow shop scheduling problem with total flow time criteria, *Int. J. Ind. Eng. Prod. Res.* 23 (2012) 301–308.
- [31] T. Stützle, Applying iterated local search to the permutation flowshop problem, Technical Report, AIDA-98-04, FG Intellektik, TU Darmstadt, 1998.
- [32] I. Osman, C. Potts, Simulated annealing for permutation flow shop scheduling, *Omega* 17 (1989) 551–557.
- [33] F. Ogbu, D. Smith, The application of the simulated annealing algorithm to the solution of the $n/m/C_{\max}$ flowshop problem, *Comput. Oper. Res.* 17 (1990) 243–253.
- [34] S.W. Lin, K.C. Ying, Minimizing makespan and total flowtime in permutation flowshops by a bi-objective multi-start simulated-annealing algorithm, *Comput. Oper. Res.* (2011), <http://dx.doi.org/10.1016/j.cor.2011.08.009>.
- [35] M. Widmer, A. Hertz, A new heuristic method for the flow shop sequencing problem, *Eur. J. Oper. Res.* 41 (1989) 186–193.
- [36] C.R. Reeves, Improving the efficiency of tabu search for machine sequencing problems, *J. Oper. Res. Soc.* 44 (1993) 375–382.
- [37] E. Nowicki, C. Smutnicki, A fast tabu search algorithm for the permutation flowshop problem, *Eur. J. Oper. Res.* 91 (1996) 160–175.
- [38] J.P. Watson, L. Barbulessu, L.D. Whitley, A.E. Howe, Contrasting structured and random permutation flowshop scheduling problems: search space topology and algorithm performance, *ORSA J. Comput.* 14 (2002) 98–123.
- [39] J. Grabowski, M. Wodecki, A very fast tabu search algorithm for the permutation flowshop problem with makespan criterion, *Comput. Oper. Res.* 31 (2004) 1891–1909.
- [40] G.I. Zobolas, C.D. Tarantilis, G. Ioannou, Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm, *Comput. Oper. Res.* 36 (2009) 1249–1267.
- [41] Y.F. Liu, S.Y. Liu, A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem, *Appl. Soft Comput.* 13 (2013) 1459–1463.
- [42] S.R. Hejazi, S. Saghafian, Flowshop scheduling problems with makespan criterion: a review, *Int. J. Prod. Res.* 43 (2005) 2895–2929.
- [43] R. Ruiz, C. Maroto, A comprehensive review and evaluation of permutation flowshop heuristics, *Eur. J. Oper. Res.* 165 (2005) 479–494.
- [44] E. Taillard, Benchmarks for basic scheduling problems, *Eur. J. Oper. Res.* 64 (1993) 278–285.
- [45] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, *Ann. Discrete Math.* 5 (1979) 287–326.
- [46] M. Dorigo, L.M. Gambardella, Ant colony system: a cooperative learning approach to the travelling salesman problem, *IEEE Trans. Evol. Comput.* 1 (1997) 53–66.
- [47] J. Grabowski, J. Pepera, New block properties for the permutation flow shop problem with application in tabu search, *J. Oper. Res. Soc.* 52 (2001) 210–220.
- [48] F. Jin, S.J. Song, C. Wu, An improved version of the NEH algorithm and its application to large-scale flow-shop scheduling problems, *IIE Trans.* 39 (2007) 229–234.
- [49] Y.R. Tzeng, C.L. Chen, C.L. Chen, A hybrid EDA with ACS for solving permutation flow shop scheduling, *Int. J. Adv. Manuf. Technol.* 60 (2012) 1139–1147.
- [50] E. Nowicki, C. Smutnicki, Some aspects of scatter search in the flow-shop problem, *Eur. J. Oper. Res.* 169 (2006) 654–666.
- [51] C.L. Chen, V.R. Neppalli, N. Aljaber, Genetic algorithms applied to the continuous flow shop problem, *Comput. Ind. Eng.* 30 (1996) 919–929.