Soft Comput (2014) 18:2271–2282
DOI 10.1007/s00500-013-1199-z

METHODOLOGIES AND APPLICATION

A revised discrete particle swarm optimization algorithm for permutation flow-shop scheduling problem

Chun-Lung Chen · Shin-Ying Huang · Yeu-Ruey Tzeng · Chuen-Lung Chen

Published online: 17 December 2013

Abstract This research proposes a revised discrete particle swarm optimization (RDPSO) to solve the permutation flow-shop scheduling problem with the objective of minimizing makespan (PFSP-makespan). The candidate problem is one of the most studied NP-complete scheduling problems. RDPSO proposes new particle swarm learning strategies to thoroughly study how to properly apply the global best solution and the personal best solution to guide the search of RDPSO. A new filtered local search is developed to filter the solution regions that have been reviewed and guide the search to new solution regions in order to keep the search from premature convergence. Computational experiments on Taillard's benchmark problem sets demonstrate that RDPSO significantly outperforms all the existing PSO algorithms.

Keywords Permutation flow-shop scheduling problem · Particle swarm optimization · Makespan

Communicated by F. Marcelloni.

Chun-Lung Chen (✉)
Department of Accounting Information, Takming University of Science and Technology, Taipei 114, Taiwan, Republic of China
e-mail: charleschen@takming.edu.tw

S.-Y. Huang
Research Center for Information Technology Innovation, Academia Sinica, Taipei 115, Taiwan, Republic of China
e-mail: smichelle19@citi.sinica.edu.tw

Y.-R. Tzeng
Department of Computer Science and Information Engineering, Chinese Culture University, Taipei 111, Taiwan, Republic of China
e-mail: ZYR3@ulive.pccu.edu.tw

Chuen-Lung Chen
Department of Management Information Systems, National Chengchi University, Taipei 116, Taiwan, Republic of China
e-mail: chen6200@gmail.com

1 Introduction

This research proposes a revised discrete particle swarm optimization (RDPSO) for the minimization of makespan in permutation flow-shop scheduling problems (PFSP-makespan). The candidate problem determines the best sequence of n jobs that are to be processed on m machines in the same order to minimize the completion time of the last job on the last machine (makespan), and has been proved to be one of the most studied NP-complete scheduling problems (Garey et al. 1976). Therefore, the development of approximate algorithms such as metaheuristics has been adopted to solve PFSP-makespan; these include: simulated annealing (SA) (Osman and Potts 1989; Ogbu and Smith 1990), tabu search (TS) (Nowicki and Smutnicki 1996; Grabowski and Wodecki 2004), genetic algorithms (GA) (Reeves 1995; Murata et al. 1996; Etiler et al. 2004; Chen et al. 2012), ant colony optimization (ACO) (Ying and Liao 2004; Rajendran and Ziegler 2004), discrete differential evolution algorithm (DDE) (Pan et al. 2008b), particle swarm optimization (PSO) (Lian et al. 2008; Zhang et al. 2010a, b; Marinakis and Marinaki 2013), and bee colony algorithm (Pan et al. 2011).

Particle swarm optimization was proposed by Kennedy and Eberhart in 1995 (Kennedy and Eberhart 1995). It imitates the behavior of a swarm of birds searching for food. The searching process of PSO for an optimization problem starts with a population of randomly generated solutions (particles; the positions of the birds in the solution space). Applying the swarm learning strategy, each particle in the population searches the solution space by considering the effect of the best solution that all the particles have ever searched (global best) and the effect of the best solution that the particle itself has ever searched (personal best). The new position of a particle in the next population is determined by its current position plus the effect of the global best solution and the effect of the personal best. This process will continue until a termination criterion is satisfied.

There have been many PSO related algorithms proposed to solve PFSP-makespan recently, and they will be discussed briefly below. $DPSO_{Ram}$ (Rameshkumar et al. 2005) and SPSOA (Zhigang et al. 2006) are two earlier PSO algorithms proposed to solve PFSP-makespan. Both of the algorithms were shown to outperform a basic GA. PSO_{vns} (Tasgetiren et al. 2007) was developed by embedding the variable neighborhood search (VNS) in a PSO algorithm. The computational results showed that PSO_{vns} dominated two ant colony algorithms, M-MMAS and PACO. H-CPSO (Jarboui et al. 2008) is a hybrid heuristic incorporating an idea of simulated annealing in a PSO algorithm. The computational results showed that H-CPSO outperformed PSO_{vns}. A discrete PSO version called NPSO (Lian et al. 2008) was developed and successfully applied to the candidate problem with results also proving to be more effective than a basic GA. Another discrete PSO ($DPSO_{Pon}$) (Ponnambalm et al. 2009) was presented and was shown to outperform ACO. HPSO (Kuoa et al. 2009) is a continuous version of PSO which integrates the random-key (RK) encoding scheme and the individual enhancement (IE) scheme into PSO. The experimental results showed that HPSO was superior to a basic GA and NPSO. ATPPSO (Zhang et al. 2010a) was developed through the integration of PSO with genetic operators and an annealing strategy. The results showed that both the solution quality and the convergence speed of ATPPSO were improved when compared to NPSO. Zhang et al. (2010b) proposed a circular discrete particle swarm optimization algorithm (CDPSO). The particle similarity changes adaptively with the iterations, and an order based strategy is introduced to preserve the swarm diversity. If the adjacent particles' similarity is bigger than its current similarity threshold, the mutation operator is used to mutate the inferior particle. Furthermore, a fast makespan computation method based on matrix is designed to improve the efficiency of the algorithm. The result showed that the solution quality and the stability of CDPSO are superior to both GA and SPSOA. Wang and Tang (2012) developed a discrete PSO using self-adaptive diversity control strategy for PFSP with blocking. They also applied their algorithm to solve PFSP-makespan and showed that their algorithm outperformed $DPSO_{Pon}$ and $DPSO_{Ram}$. Marinakis and Marinaki (2013) presented a PSO with expanding neighborhood topology (PSOENT). The major difference between PSOENT and the aforementioned PSO algorithms is that PSOENT does not use local search methods. It combines a PSO algorithm, a VNS strategy, and a path relinking strategy. Computational results showed that PSOENT outperforms all the PSO algorithms without using local search methods.

RDPSO proposes new particle swarm learning strategies to thoroughly study how to properly apply the global best solution and the personal best solution to guide the search of RDPSO. A new filtered local search (FLS) is developed to filter the solution regions that have been reviewed and guide the search to new solution regions in order to keep the search from premature convergence. Computational experiments on Taillard's benchmark problem sets (Taillard 1993) will be performed to evaluate the effectiveness of RDPSO by comparing its performance with several state-of-the-art PSO heuristics and DDE_{RLS} (Pan et al. 2008b), the most effective heuristic for PFSP-makespan up to now. The remainder of the paper is organized as follows: Sect. 2 presents the proposed RDPSO. Section 3 provides computational experiments, and conclusions and future works of this study are summarized in Sect. 4.

2 Proposed RDPSO algorithm

Particle swarm optimization was proposed by Kennedy and Eberhart in 1995 (Kennedy and Eberhart 1995). It imitates the behavior of a swarm of birds searching for food. The standard PSO equations for updating positions for birds are real-valued equations; therefore, discrete PSO (DPSO) algorithms (Pan et al. 2008a) have been developed to solve PFSP-makespan. The main components of DPSO include population initialization, position update for particles and a local search for improving the solution quality. A discrete position update equation can be expressed as follows:

$$X_i^t = c_2 \otimes F_3(c_1 \otimes F_2(w \otimes F_1(X_i^{t-1}), P_i^{t-1}), G^{t-1}) \quad (1)$$

Given that the position of particle i in iteration $t-1$ is X_i^{t-1}, this equation first implements function F_1 with a probability of w; function F_1 searches the neighborhood of X_i^{t-1}. Then the equation implements function F_2 with a probability of c_1. Function F_2 exchanges information with the solution generated by function F_1 and the personal best solution of particle i (P_i^{t-1}); it refers to the condition that particle i will learn from its personal best solution. Finally, this equation implements function F_3 with a probability of c_2. Function F_3 exchanges information with the solution generated by functions F_2 and the global best solution; it refers to the condition that particle i will learn from the global best solution.

In this research, we alter the DPSO algorithm to develop RDPSO. New particle swarm learning strategies are proposed to update particle position in order to guide the search of RDPSO. A new FLS is developed to filter the solution regions that have been reviewed and guide the search to new solution regions in order to keep the search from premature convergence. The flowchart of RDPSO is presented in Fig. 1. It first randomly generates the initial population with P_{num} particles. The new particle swarm learning strategies are then applied to update the positions of the particles. Three learning parameters, c_1, c_2 and w, are used to determine the probabilities that a particle will learn from the global best

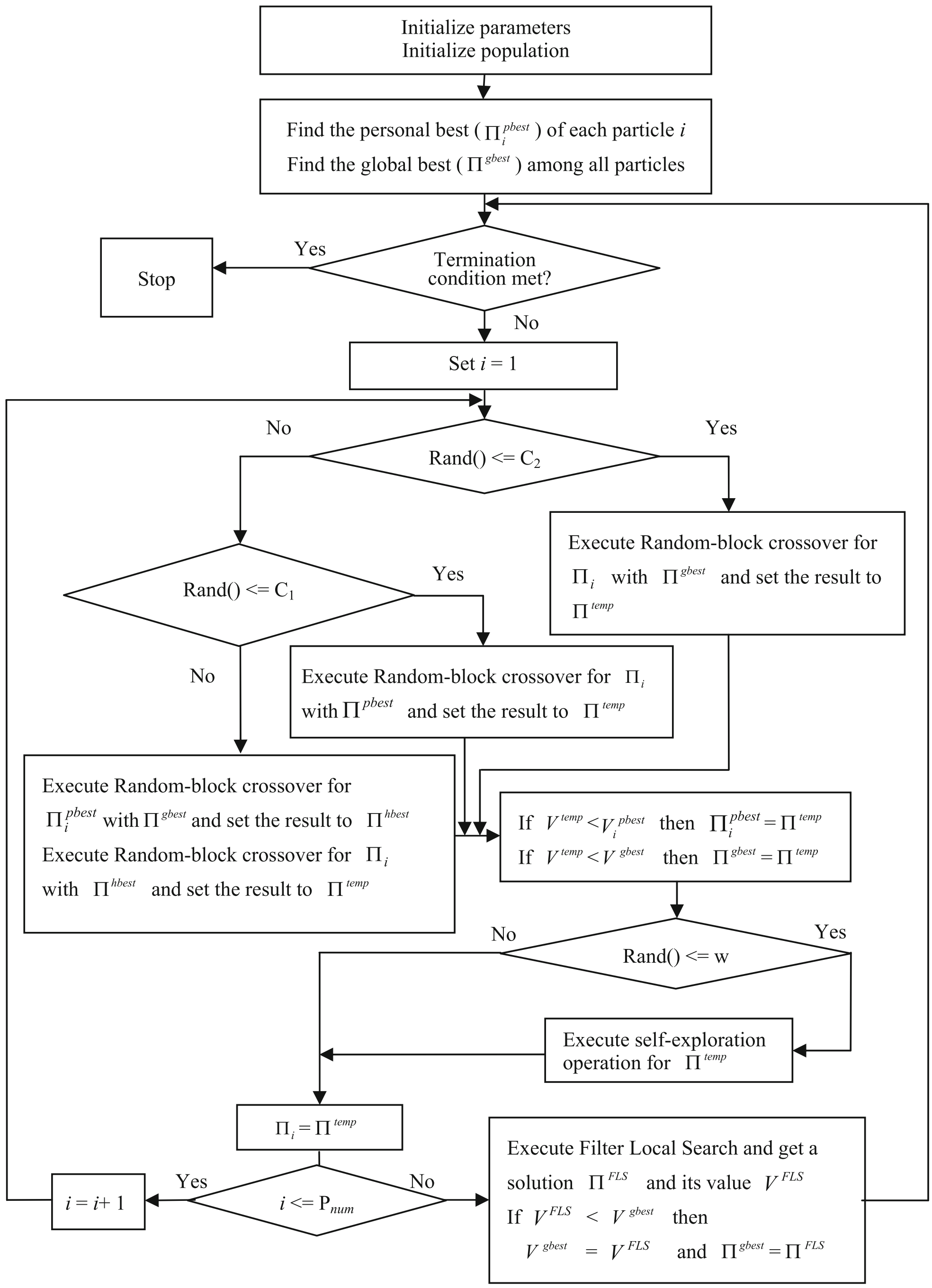

Fig. 1 The flowchart for RDPSO

solution, its personal best solution and searching its neighborhood. When the positions of all the particles are updated, the new filtered local search will be implemented. This searching process will continue until a termination criterion is satisfied.

The details of the new particle swarm learning strategies and the new filtered local search are discussed in the following subsections and the notations used in Fig. 1 are described as follows: Π_i is the solution of particle i; V_i^{pbest} is the objective value of the personal best solution of particle i (Π_i^{pbest}); V^{gbest} is the objective value of the global best solution (Π^{gbest}); V_i^{hbest} is the objective value of the hybrid best solution of particle i (Π_i^{hbest}); V^{temp} is the objective value of a temporary solution (Π^{temp}); P_{num} is the number of particles generated in an iteration; and V^{FLS} is the objective value of the solution (Π^{FLS}) generated by the filtered local search. The hybrid best solution of particle i (Π_i^{hbest}) is constructed by applying a crossover operation to the personal best solution of particle i ($\prod_i^{pbest}$) and the global best solution (Π^{gbest}).

Table 1 The six combinations of c_1 and c_2 considered in the first phase of the new particle swarm learning strategies

c_1	c_2	p_g	p_p	p_h
0.9	0.1	0.1	0.81	0.09
0.9	0.3	0.3	0.63	0.07
0.9	0.5	0.5	0.45	0.05
0.9	0.7	0.7	0.27	0.03
0.9	0.9	0.9	0.09	0.01
0.1	0.1	0.1	0.09	0.81

Table 2 The 12 F-Strategies used in the new particle swarm learning strategies

F-Strategy	First 500 iterations		Second 500 iterations	
	p_g	p_p	p_g^*	p_p^*
F-Strategy 1	0.1	0.81	0.1	0.81
F-Strategy 2	0.1	0.81	0.81	0.1
F-Strategy 3	0.3	0.63	0.3	0.63
F-Strategy 4	0.3	0.63	0.63	0.3
F-Strategy 5	0.5	0.45	0.5	0.45
F-Strategy 6	0.5	0.45	0.45	0.5
F-Strategy 7	0.7	0.27	0.7	0.27
F-Strategy 8	0.7	0.27	0.27	0.7
F-Strategy 9	0.9	0.09	0.9	0.09
F-Strategy 10	0.9	0.09	0.09	0.9
F-Strategy 11	0.1	0.09	0.1	0.09
F-Strategy 12	0.1	0.09	0.09	0.1

2.1 The new particle swarm learning strategies

Several DPSO algorithms (Kuoa et al. 2009; Zhang et al. 2010a, b) have been developed for solving PFSP-makespan. Most of them follow the learning strategy of Eq. (1), that is a particle will start learning from searching its neighborhood (function F_1), then from its personal best solution (function F_2), and then from the global best solution (function F_3). Since both of the parameters, c_1 and c_2, in the equation are less than or equal to1.0, they may cause F_2 and F_3 to not be implemented, and accordingly, a particle will not learn from its personal best solution and the global best solution.

This research proposes new learning strategies that guarantee a particle to learn from its personal best solution or the global best solution. The learning strategy is separated into two phases. In the first phase, a particle will learn sequentially first from the global best solution, then its personal best solution, and finally its hybrid best solution. In the second phase, the particle will search its neighborhood. (In this research, the proposed learning strategy is denoted as two-phase strategy, and the strategy following equation is denoted as single-phase strategy.) The values of the parameters (c_1, c_2 and w) are properly determined in order to generate the probabilities that a particle will learn from the global best solution (p_g), its personal best solution (p_p), and its hybrid best solution (p_h) in the first phase, and the particle will search its neighborhood with probability (w) in the second phase. Table 1 presents six combinations of c_1 and c_2 that are considered in the first phase. The p_g, p_p, and p_h values for each of the combinations are calculated following the flow chart in Fig. 1. It will first test c_2 to determine p_g ($p_g = c_2$), then it will test c_1 to determine p_p ($p_p = (1 - c_2) \times c_1$) and the rest of the probability is the probability of $p_h (p_h = 1 - p_g - p_p)$. For instance, if $c_1 = 0.9$ and $c_2 = 0.1$ (the first combination in Table 1), then $p_g = c_2 = 0.1$, $p_p = (1 - c_2) \times c_1 = 0.81$ and $p_h = 1 - p_g - p_p = 0.09$. The p_g, p_p, and p_h values for the combinations in Table 1 demonstrate the purpose of selecting the six combinations. A combination with smaller c_2 (when $c_1 = 0.9$) will have a smaller p_g and a larger p_p; it infers that, under this condition, a particle will learn more from its personal best solution. On the contrary, a combination with larger c_2 (when $c_1 = 0.9$) will have a larger p_g and a smaller p_p, and a particle will learn more from the global best solution under this condition. The last combination ($c_1 = 0.1$ and $c_2 = 0.1$) is designed for the condition that a particle will particularly learn from its hybrid best solution ($p_h = 0.81$). In addition, the six levels of w are 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0.

Another idea is further considered for the learning strategies. Given a combination in Table 1, a learning strategy using the same p_g and p_p of the combination in the whole searching process is denoted as a fixed learning strategy; a strategy using the p_g and p_p in the first half of the searching process and using $p_g^*(=p_p)$ and $p_p^*(=p_g)$ in the second half of the searching process is denoted as a variable learning strategy. For instance, if a learning strategy employs the p_g and p_p values of the first combination in Table 1 and 1,000 iterations are used in the searching process, then a variable learning strategy searches the solution space using $p_g = 0.1$ and $p_p = 0.81$ in the first 500 iterations and using $p_g = 0.81$ and $p_p = 0.1$ in the second 500 iterations. The motivation of using the variable learning strategy is to overcome certain limitations present with a fixed learning strategy. When a fixed learning strategy is applied with a large p_g value, particles will learn mostly from the global best solution and may cause the search to quickly trap into local optima. Furthermore, when a fixed strategy is applied with a large p_p value, particles will learn mostly from the personal best solution and may keep the search from converging into good solution regions. Therefore, it is speculated that a variable learning strategy with proper p_g and p_p values may enhance the performance of RDPSO. Table 2 summarizes 12 different strategies used in the first phase, denoted as F-Strategy, of the particle swarm learning strategies.

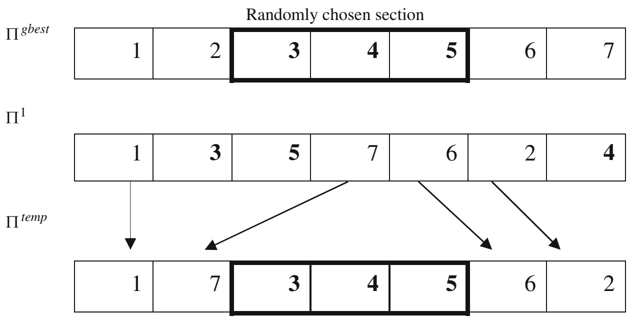

Fig. 2 An example for the random-block crossover

The learning operation of a particle from a chosen solution is executed by applying a random-block crossover operation to the particle and the chosen solution. Given Π^i is the solution of particle i and Π^{gbest} is the chosen solution, applying the random block crossover operation to Π^i and Π^{gbest} includes three steps. The first step is to randomly choose a consecutive job sequence, a random block, from Π^{gbest}. The size of the random block is equal to Int($B \times n$), where Int() is the integer function, B is a predefined real value between 0.0 and 1.0, and n is the number of the jobs considered in a solution. The second step is to construct a new solution by assigning the random block to the same position in the new solution as that in Π^{gbest}. The last step is to sequentially assign the jobs in Π^i, which is not in the random block, to the empty positions in the new solution. A simple example illustrating the crossover operation is shown in Fig. 2; let $\Pi^{gbest} = (1, 2, 3, 4, 5, 6, 7)$, $\Pi^i = (1, 3, 5, 7, 6, 2, 4)$, and $B = 0.5$, so the size of a random block is $\text{Int}(0.5 \times 7) = 3$. A block (3, 4, 5) is randomly chosen in Step 1, so the new solution constructed in Step 2 is (–, –, 3, 4, 5, –, –). The last step will then assign job 1, the first job in Π^i, to the first position in the new solution. Since job 3 and job 5, the second and the third jobs in Π^i, belong to the random block, job 7, the fourth job in Π^i, will then be assigned to the second position in the new solution, and job 6 and job 2 will be assigned to the last two positions in the new solution.

After a particle learns from its personal best solution and the global best solution in the first phase of a learning strategy, it explores its neighborhood, a self-exploration operation, in the second phase of the strategy with a probability w. The algorithm of the self-exploration operation is inspired by Ruiz and Stutzle's (2007) destruction and the construction procedures. Given a solution Π, the self-exploration operation first randomly chooses four jobs from the solution. Let the job sequence of the four jobs be Π_1 and the job sequence of the rest of the jobs be Π_2. Then, insert the first job in Π_1 into the first position, the last position and the positions between every two consecutive jobs in Π_2 and choose the sequence with the smallest makespan; repeat the same process until all the four jobs in Π_1 are inserted in Π_2.

2.2 Filtered local search

When all the particles generate their solutions in an iteration, the FLS is applied to improve the global best solution. Local search methods are crucial for improving the effectiveness of population-based metaheuristics such as DPSO (Zhang et al. 2010b; Pan et al. 2008a) and ACO (Rajendran and Ziegler 2004; Dorigo and Stützle 2004). 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 local optima. The FLS first applies a filter function to find a solution in an iteration, and then applies a local search method to the chosen solution. The purpose of the filter function is to guide the search to the solution regions which have not been examined and protect the search from trapping into local optima.

The proposed filter function is applied when all the P_{num} solutions are generated in an iteration. We define filter-list as a first-in, first-out queue to store the makespan of the solution chosen in each iteration and set a parameter called filter-size to define the size of the queue. The queue is set to be empty initially. When all the P_{num} solutions are generated in an iteration, the solutions are sorted according to their makespans in ascending order, and the filter function is applied from the top of the P_{num} solutions until the first solution, whose makespan is different from all the makespans in the filter-list, is found and store the makespan of the solution in the filter list. If none of the P_{num} solutions has a different makespan from the makespans in the filter-list, the last of the P_{num} solutions is chosen but the makespan will not store in the filter-list. The purpose of comparing makespans instead of job-sequences of solutions while using the filter function is twofold. 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 an individual and the solutions stored in the filter-list; this is especially critical when the number of jobs considered in a problem is large. The ten 50-job and 20-machine instances chosen from the well-known Taillard's test problems are used to compare the performance of the RDPSO with comparing makespan and that with comparing job-sequence. Computational results demonstrate that the RDPSO with comparing makespan on average dominates the RDPSO with comparing job-sequence by 30 % using 16 % less execution time. These findings reveal the rationale for comparing makespan while using the filter function.

Once a solution is chosen using the filter function, the local search method NEHT_LS is applied to improve the makespan of the solution. NEHT_LS integrates Taillard's Modified-NEH method (Taillard 1990) with Ruiz and Stutzle's (2007) 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 global best solution, it will update the global best solution with Π.

3 Computational experiments

The well-known Taillard's test problems for PFSP-makespan (Taillard 1993) are used to evaluate the performance of the RDPSO. The test problems are composed of 12 different problem sets with different numbers of jobs (n) and different numbers of machines (m) and ten test instances are generated in each of the 12 problem sets, so there are a total of 120 test problems. Twelve instances, selecting the first instance from each of the 12 problem sets (denoted as *Test1*), are used to investigate the effects of the three major parameters of RDPSO: the F-Strategy, the w value, and the filter-size. Then, the RDPSO with the best combination of the three parameters are applied to solve twenty-eight problems, which are chosen from the 12 problem sets [NPSO (Lian et al. 2008), ATPPSO (Zhang et al. 2010a) and CDPSO (Zhang et al. 2010a)], denoted as *Test2*, and to solve all the test problems except the problems with 500 jobs [PSO_{vns} (Tasgetiren et al. 2007) and H-CPSO (Jarboui et al. 2008)], denoted as *Test3*, in order to compare its performance with existing promising PSO algorithms. In addition, DDE_{RLS} (Pan et al. 2008b), the most effective heuristic for PFSP-makespan up to now, is used to further evaluate the performance of RDPSO.

As mentioned in Tables 1 and 2, 12 levels of F-Strategy and six levels of w are considered in the first and the second phases of the swarm learning strategy respectively. The filter-size has two levels and is set to 0 and 7, where 0 refers to that which no filter function is applied. Therefore, there are a total of 144 different combinations of the three major parameters. The remaining parameters of RDPSO are described as follows: the population size is set to be 60; the block size for the random-block crossover is set to be (3/20) $\times n$ and the termination criterion is set to be 1000 generations. All these parameters are determined by trial-and-error.

The RDPSO with each of the 144 combinations is then applied to solve the 12 instances in $Test_1$ for three trials. The average relative performance (ARP) is used to measure the performance of the RDPSO with a combination for each instance. The formula of ARP is as follows: $ARP = \sum_{i=1}^{R} (\frac{solution_i - Best_{sol}}{Best_{sol}} \times 100)/R$; given an instance, $solutionu_i$ is the makespan obtained by trial i of the RDPSO with a combination for the instance, and $Best_{sol}$ is the best makespan that all the research has found for the instance provided by Zobolas et al. (2009).

Table 3 ANOVA table for testing the significance of the major parameters of RDPSO

Source	Type III sum of squares	df	Mean square	F	Sig.
Corrected model	1,195.533(a)	28	42.698	770.531	0.000
Intercept	1,070.424	1	1,070.424	19,317.134	0.000
F-Strategy	2.35	11	0.214	3.855	0.000
w value	50.661	5	10.132	182.847	0.000
Filter-size	6.579	1	6.579	118.722	0.000
Instance	1,135.944	11	103.268	1,863.592	0.000
Error	94.147	1,699	0.055		
Total	2,360.104	1,728			
Corrected total	1,289.68	1, 727			

a R Squared = 0.927(Adjusted R Squared = 0.926)

Table 4 Result of the RDPSO with filter-size = 0 and filter-size = 7

Filter-size	Mean	Std. error	95 % confidence interval	
			Lower bound	Upper bound
0	0.849	0.008	0.833	0.864
7	0.725	0.008	0.710	0.741

Table 5 Result of the Duncan test for the F-Strategy

Method	N	Subset		
		1	2	3
F-Strategy 4	144	0.7171		
F-Strategy 2	144	0.7491	0.7491	
F-Strategy 5	144	0.7522	0.7522	
F-Strategy 6	144	0.7617	0.7617	
F-Strategy 8	144		0.7818	
F-Strategy 9	144		0.7950	
F-Strategy 10	144		0.7960	
F-Strategy 7	144		0.8036	
F-Strategy 11	144		0.8047	
F-Strategy 12	144		0.8067	
F-Strategy 3	144		0.8091	
F-Strategy 1	144			0.8677

Table 6 Result of the Duncan test for the w value

w	N	Subset		
		1	2	3
0.6	288	0.6717		
0.8	288	0.6932		
1.0	288	0.6965		
0.4	288	0.6988		
0.2	288		0.8042	
0.0	288			1.1579

Table 7 *Paired-t* test for single-phase strategy and two-phase strategy

		Paired differences					t	df	Sig. (2-tailed)
		Mean	Std. deviation	Std. error mean	95 % confidence interval of the difference				
					Lower	Upper			
Pair 1	Single-phase strategy–two-phase strategy	0.0683	0.0994	0.0287	0.0051	0.1315	2.380	11	0.036

The analysis of variance (ANOVA) is applied to analyze the ARPs produced by the RDPSO with all the 144 combinations for the instances in $Test_1$. Table 3 presents the results of the ANOVA table (generated by using SPSS). The results show that all the three major parameters, the F-Strategy, the w value, and the filter-size, significantly affect the performance of RDPSO. Since the filter-size considers only two levels, 0 and 7, the results of ANOVA have shown that the performance of the RDPSO using these two levels is significantly different. Table 4 presents the average ARPs produced by the RDPSO with filter-size = 0 and filter size = 7; the results show that the RDPSO using filter strategy (filter size = 7) significantly dominates the RDPSO without using filter strategy (filter size = 0). The Duncan's multiple range test is then applied to determine if the performance of any two levels of the F-Strategy and the w value is significantly different. Tables 5 and 6 present the results of the Duncan's test for the F-Strategy and the w value respectively. Note that the levels of the F-Strategy and the w value in Tables 5 and 6 are sequenced in ascending order in terms of their mean ARPs, and the levels in the same subset represent that the performance of the RDPSO with the levels is not significantly different.

Table 8 ARPs of RDPSO, CDPSO, ATPPSO, and NPSO

Problem	Size	Optimal	CDPSO	ATPPSO	NPSO	RDPSO
ta001	20×5	1,278	0.59	0.00	1.33	0.00
ta011	20×10	1,582	0.29	0.03	1.43	0.00
ta015	20×10	1,419	0.31	0.23	0.64	0.07
ta021	20×20	2,297	0.45	0.30	1.21	0.00
ta025	20×20	2,291	0.17	0.28	0.92	0.17
ta031	50×5	2,724	0.15	0.01	0.20	0.00
ta035	50×5	2,863	0.03	0.02	0.03	0.00
ta040	50×5	2,782	0.00	0.01	0.04	0.00
ta041	50×10	2,991	2.44	2.47	3.59	1.48
ta045	50×10	2,976	2.09	2.22	3.45	1.16
ta051	50×20	3,771–3,847	2.50	2.32	3.75	1.28
ta055	50×20	3,553–3,610	2.74	2.83	4.53	1.12
ta061	100×5	5,493	0.00	0.00	0.01	0.00
ta065	100×5	5,250	0.08	0.07	0.12	0.02
ta071	100×10	5,770	0.69	0.71	1.26	0.19
ta075	100×10	5,467	1.31	1.41	2.03	0.62
ta081	100×20	6,106–6,202	3.43	–	–	1.73
ta085	100×20	6,262–6,314	3.19	3.27	5.37	1.57
ta090	100×20	6,404–6,434	2.56	2.65	4.49	1.53
ta091	200×10	10,862	0.65	0.53	1.07	0.17
ta095	200×10	10,524	0.39	0.36	1.33	0.12
ta100	200×10	10,675	0.69	0.65	1.14	0.21
ta101	200×20	11,152–11,181	3.25	3.29	4.21	1.54
ta105	200×20	11,259	2.40	2.80	4.12	0.98
ta110	200×20	11,284–11,288	3.25	3.71	4.82	1.42
ta111	500×20	26,040–26,059	2.51	2.78	3.68	0.71
ta115	500×20	26,334	2.27	1.97	2.98	0.46
ta120	500×20	26,457	1.81	2.11	3.11	0.60
average			1.44	1.37	2.25	0.61

– This data are not generated by ATPPSO and NPSO

The results in Table 5 show several valuable findings. First, F-Strategy 4 produces the best mean ARP; it significantly dominates its corresponding fixed strategy, F-Strategy 3, which produces the second worst mean ARP. Also, F-Strategy 2 produces the second best mean APR; it significantly dominates its corresponding fixed strategy, F-Strategy 1, which produces the worst mean ARP. Note that the (p_g, p_p) values of F-Strategy 3 and F-Strategy 1 are (0.3, 0.63), (0.1, 0.81) respectively. This result confirms our conjecture that the RDPSO using fixed F-Strategy with high p_p value may not be able to converge to good solution regions. On the contrary, the RDPSO using variable F-Strategy with higher p_p value in the first 500 iterations and higher p_g value in the second 500 iterations of the search may take advantages of exploration in the first 500 iterations and exploitation in the second 500 iterations and converge to promising solution regions. Second, the RDPSO using F-Strategy 2 significantly dominates the RDPSOs using F-Strategy 8 and F-Strategy 10. The (p_g, p_p) values of F-Strategy 8 and F-Strategy 10 are (0.7, 0.27), (0.9, 0.09) respectively. These finding illustrate that although F-Strategy 2, F-Strategy 8, and F-Strategy 10 are all variable strategies, the RDPSO, using larger p_p value in the first 500 iterations, may have better exploration capability and lead the search to better solution regions. Third, the RDPSOs using F-Strategy 11 and F-Strategy 12 produce poor mean ARP; the (p_g, p_p) value of F-Strategy 11 is (0.1,

Table 9 *Paired-t* test for RDPSO, CDPSO, ATPPSO, and NPSO

		Paired differences					t	df	Sig. (2-tailed)
		Mean	Std. deviation	Std. error mean	95 % confidence interval of the difference				
					Lower	Upper			
Pair 1	CDPSO-RDPSO	0.8333	0.6687	0.1287	0.5688	1.0979	6.476	26	0.000
Pair 2	ATPPSO-RDPSO	0.8312	0.7527	0.1476	0.5271	1.1352	5.631	25	0.000
Pair 3	NPSO-RDPSO	1.6965	1.2087	0.2371	1.2083	2.1848	7.156	25	0.000

Table 10 Average ARPs of PSO_{vns}, H-CPSO, DDE_{RLS} and RDPSO

Problem set	PSOvns	H-CPSO	DDE_{RLS}	RDPSO
20×5	0.03	0.00	0.04	0.00
20×10	0.02	0.01	0.01	0.01
20×20	0.05	0.02	0.02	0.01
50×5	0.00	0.00	0.00	0.00
50×10	0.57	0.49	0.45	0.49
50×20	1.36	0.96	0.66	0.83
100×5	0.00	0.02	0.00	0.00
100×10	0.18	0.26	0.15	0.15
100×20	1.45	1.28	0.98	1.24
200×10	0.18	0.40	0.07	0.12
200×20	1.35	1.55	0.99	1.38
Overall average ARP	0.47	0.45	0.31	0.39

0.09) and its p_h is 0.81. This result concludes that the RDPSO using both low p_g and p_p values will not converge to good solution regions.

The results in Table 6 show that w values, 0.6, 0.8, 1.0, and 0.4, belong to the first subset. This result illustrates that the RDPSO should use at least a w value of 0.4 in the second phase of the learning strategy. The mean ARP show that the RDPSO using $w = 0.6$ dominates the RDPSO using $w = 0.2$ and $w = 0.0$ by 16 % ((0.8042–0.6717)/0.8042)) and 42 % ((1.1579–0.6717)/1.1579)) respectively. The previous analyses conclude that the best parameter set for the RDPSO is F-Strategy = F-Strategy 4, $w = 0.6$ and filter-size = 7.

Note that the learning strategy of RDPSO is a two-phase strategy, and the learning strategy of most of the DPSO algorithms, using the learning sequence of Eq. (1), is a single-phase strategy. Pan et al. considered two levels (0.2, 0.8) for each of w, c_1 and c_2 of Eq. (1) to determine an appropriate parameter set for their DPSO. In order to investigate the effect of the two-phase strategy, we replace the two-phase strategy in RDPSO with the single-phase strategy and execute the modified RDPSO with each of the eight combinations used by Pan et al. to solve the 12 instances in $Test_1$ for three trials. The results show that the best parameter set for the modified RDPSO is $w = 0.8$, $c_1 = 0.2$ and $c_2 = 0.8$. The *paired-t* test is then applied to test the significance of the difference between the performance of RDPSO using the single-phase strategy ($w = 0.8$, $c_1 = 0.2$ and $c_2 = 0.8$) and RDPSO using the two-phase strategy (F-Strategy = F-Strategy 4, $w = 0.6$). Table 7 presents the results of the *paired-t* test and demonstrates that RDPSO using the two-phase strategy significantly outperforms RDPSO using the single-phase strategy.

The RDPSO with the best parameter set is then applied to solve the test problems in *Test2*. Ten trials are implemented for each heuristic for each test problem. Note that all the algorithms use 1000 iterations as the termination criterion. Table 8 presents the ARPs generated by NPSO, CDPSO, ATPPSO and RDPSO. The notation used in the first column of the table denotes the problem number of the twenty-eight problems in the 120 test problems (Taillard 1990), which are denoted from ta001 to ta120. The results show that RDPSO outperforms NPSO, CDPSO, and ATPPSO in all the test problems. The average ARP show that RDPSO dominates

Table 11 *Paired-t* test for PSO_{vns}, H-CPSO, DDE_{RLS} and RDPSO

		Paired differences					t	df	Sig. (2-tailed)
		Mean	Std. deviation	Std. error mean	95 % confidence interval of the difference				
					Lower	Upper			
Pair 1	PSOvns-RDPSO	0.08644	0.15985	0.04820	−0.2095	0.19383	1.793	10	0.103
Pair 2	H-CPSO-RDPSO	0.06825	0.09270	0.02795	0.00598	0.13053	2.442	10	0.035
Pair 3	DDE_{RLS}−RDPSO	−0.07818	0.13681	0.04125	−0.1701	0.01373	−1.8954	10	0.09

CDPSO by 58 % ((1.44–0.61)/1.44), dominates ATPPSO by 55 % ((1.37–0.61)/1.37), and dominates NPSO by 73 % ((2.25–0.61)/2.25). Furthermore, the *paired-t* test is applied to test the significance of the difference between the performance of RDPSO and the performance of each of CDPSO, ATPPSO and NPSO for the test problems. Table 9 presents the results of the *paired-t* test and shows that RDPSO significantly dominates CDPSO, ATPPSO and NPSO.

In addition, the RDPSO with the best parameter set is applied to solve the test problems in *Test3*, and its performance is compared to the computational results produced by two promising PSO algorithms (PSO_{vns} and H-CPSO) and the most effective algorithm for PFSP-makespan, DDE_{RLS}. PSO_{vns} solved the problems using a PC with an Intel Pentium IV at 2.6 GHz, and it solved each problem ten times to calculate the ARP. The termination criterion was determined

Table 12 Solutions generated by the RDPSO without local search and PSOENT

Problem	PSOENT	RDPSO	Problem	PSOENT	RDPSO	Problem	PSOENT	RDPSO
20×5	**1278**	**1278**	50×10	3092	**3051**	100×20	6430	**6414**
20×5	**1359**	**1359**	50×10	2942	**2915**	100×20	6489	**6383**
20×5	**1081**	**1081**	50×10	2926	**2889**	100×20	6526	**6437**
20×5	**1293**	**1293**	50×10	3083	**3071**	100×20	6440	**6407**
20×5	**1235**	**1235**	50×10	3049	**3024**	100×20	6612	**6509**
20×5	**1195**	**1195**	50×10	3056	**3036**	100×20	6633	**6551**
20×5	**1239**	**1239**	50×10	3144	**3133**	100×20	6605	**6476**
20×5	**1206**	**1206**	50×10	3072	**3049**	100×20	6724	**6640**
20×5	**1230**	**1230**	50×10	2952	**2923**	100×20	6576	**6462**
20×5	**1108**	**1108**	50×10	3143	**3131**	100×20	6699	**6593**
20×10	**1582**	**1582**	50×20	4004	**3950**	200×10	10953	**10872**
20×10	**1659**	**1659**	50×20	3838	**3761**	200×10	10610	**10556**
20×10	1500	**1496**	50×20	3788	**3741**	200x10	11040	**10950**
20×10	**1377**	**1377**	50×20	3857	**3806**	200×10	10939	**10893**
20×10	**1419**	**1419**	50×20	3732	**3688**	200×10	10646	**10537**
20×10	**1397**	**1397**	50×20	3821	**3758**	200×10	10452	**10378**
20×10	**1484**	**1484**	50×20	3855	**3763**	200×10	10977	**10882**
20×10	1544	**1543**	50×20	3825	**3788**	200×10	10864	**10777**
20×10	**1593**	**1593**	50x20	3903	**3831**	200×10	10498	**10450**
20×10	**1591**	1598	50×20	3896	**3830**	200×10	10810	**10727**
20×20	2298	**2297**	100×5	**5493**	**5493**	200×20	11571	**11535**
20×20	2101	**2100**	100×5	5274	**5268**	200×20	11729	**11596**
20×20	2328	**2326**	100×5	5179	**5175**	200×20	11757	**11676**
20×20	2225	**2223**	100×5	5023	**5014**	200×20	11713	**11665**
20×20	**2294**	**2294**	100×5	5255	**5250**	200×20	11712	**11548**
20×20	2229	**2228**	100×5	**5135**	**5135**	200×20	11699	**11546**
20×20	**2273**	**2273**	100×5	5251	**5246**	200×20	11874	**11702**
20×20	2202	**2200**	100×5	**5094**	**5094**	200×20	11813	**11675**
20×20	2240	**2237**	100×5	5454	**5448**	200×20	11725	**11554**
20×20	**2178**	**2178**	100×5	5332	**5322**	200×20	11780	**11683**
50×5	**2724**	**2724**	100×10	5851	**5790**	500×20	26737	**26656**
50×5	2838	**2836**	100×10	5407	**5377**	500×20	27497	**27153**
50×5	**2621**	**2621**	100×10	5691	**5679**	500×20	27277	**26923**
50×5	**2751**	**2751**	100×10	5902	**5849**	500×20	27080	**26894**
50×5	**2863**	**2863**	100×10	5588	**5514**	500×20	26915	**26768**
50×5	**2829**	**2829**	100×10	5334	**5308**	500×20	27203	**26965**
50×5	**2725**	**2725**	100×10	5658	**5602**	500×20	27057	**26799**
50×5	**2683**	**2683**	100×10	5695	**5664**	500×20	27270	**27066**
50×5	**2554**	2555	100×10	5958	**5907**	500×20	26622	**26488**
50×5	**2782**	**2782**	100×10	5903	**5857**	500×20	27164	**26923**

via the execution-time and was set at 10 min for problems with 100 jobs or more and 5 min for the rest of the problems. H-CPSO solved the problems using a PC with Intel Pentium IV at 3.2 GHz, and it solved each problem five times and terminated at an execution-time of 500 s for problems with 100 jobs or more and 250 s for the rest of the problems. RDPSO solved the problems using a PC with Intel Pentium IV at 2.8 GHz and utilized the same computational conditions as H-CPSO. DDE_{RLS} solved the problems using a PC with Intel Pentium IV at 3.0 GHz, and it solved each problem five times and terminated at an execution-time of $((n \times m)/2{,}000) \times 30$ s. Note that n refers to the number of jobs and m refers to the number of machines in an instance. Therefore, the execution times of DDE_{RLS} are much less than those used for PSO_{vns}, H-CPSO and RDPSO. Table 10 presents the average ARPs generated by PSO_{vns}, H-CPSO, DDE_{RLS} and RDPSO for the problems in each of the eleven problem sets. Table 11 presents the results of the *paired-t* test for PSO_{vns}, H-CPSO, DDE_{RLS} and RDPSO. The results show that RDPSO dominates H-CPSO at the significance level of 0.035 and dominates PSO_{vns} at the significance level of 0.103. However, DDE_{RLS} dominates RDPSO at the significance level of 0.09.

Table 13 Average ARPs of the RDPSO without local search and PSOENT

Problem set	PSOENT	RDPSO
20 × 5	**0.00**	**0**
20 × 10	**0.07**	0.08
20 × 20	0.08	**0.03**
50 × 5	**0.02**	**0.02**
50 × 10	2.11	**1.31**
50 × 20	3.83	**2.2**
100 × 5	0.09	**0**
100 × 10	1.26	**0.48**
100 × 20	4.37	**3**
200 × 10	1.02	**0.3**
200 × 20	4.27	**3.21**
500 × 20	2.73	**1.9**
Overall average ARP	1.65	**1.04**

Finally, the performance of the RDPSO algorithm without using the local search method, NEHT_LS, is compared with the recently proposed PSO with expanding neighborhood topology (PSOENT) of Marinakis and Marinaki (2013), the best PSO without using local search methods. For equal comparison, the RDPSO without using local search and PSOENT are applied to solve all the 120 test instances with the same termination criterion of 1,000 iterations for ten trials, and the best solution of the ten trials for each instance is presented in Table 12. Note that the solutions of PSOENT in Table 12 are from Marinakis and Marinaki (2013); the better solution between the two algorithms for each instance is presented with bold. The results demonstrate that the RDPSO without using local search is superior to PSOENT in 87 instances (72.5 % of all the instances), tied with PSOENT in 31 instances (25.8 % of all the instances), and inferior to PSOENT in only 2 instances. The ARPs of the two algorithms are then calculated for each instance, and the average ARPs of the ten instances for each of the 12 problem sets are presented in Table 13. The results show that the overall average ARP of the RDPSO without using local search dominates PSOENT by 40 % ((1.654–1.04)/1.65). Table 14 presents the results of the *paired-t* test for the RDPSO without using local search and PSOENT. The result shows that RDPSO without using local search dominates PSOENT at the significance level of 0.004.

4 Conclusions

This paper proposes a revised PSO algorithm (RDPSO) to solve the PFSP-makespan. Computational experiments have shown that the proposed swarm learning strategies and the new filtered local search method significantly improve the performance of RDPSO. Also, the performance of RDPSO dominates all the existing PSO algorithms using local search. Additionally, the performance of RDPSO without using local search dominates PSOENT, the best PSO without using local search. However, RDPSO is inferior to the performance of DDE_{RLS}.

Some ideas can be further studied to improve the performance of RDPSO. Since all the existing PSO algorithms

Table 14 *Paired-t* test for RDPSO and PSOENT

		Paired differences					t	df	Sig. (2-tailed)
		Mean	Std. deviation	Std. error mean	95 % confidence interval of the difference				
					Lower	Upper			
Pair 1	PSOENT-RDPSO	0.61	0.57671	0.16648	0.09294	1.12706	3.664	11	0.004

used random initial population, RDPSO used random initial population in order to compare its performance with the existing PSO algorithms. However, DDE_{RLS} used NEH to construct its initial population and reported that the promising initial population significantly improved its performance. Therefore, it is believed that the application of NEH to RDPSO could help generate a promising initial population and enhance the performance of RDPSO. Furthermore, the constitution of the members in its initial population can be studied. For example, the initial population can be generated based on the schedule produced by NEH only (DDE_{RLS}) or it can be generated based on the schedules produced by NEH and other well known heuristics such as CDS (Sipper and Bulfin 1997). It can also be constituted with part of the solutions generated by well known heuristics and part of the solutions generated randomly. The investigation into the effect of different modes of initial population may lead to further improvements in RDPSO performance.

Another area of research to be investigated for RDPSO improvement is escape strategies. Although the new filter function is able to keep the search of RDPSO from quick convergence, RDPSO may still trap into local optima. Therefore, research towards the development of escape strategies for guiding the search to jump from a local optimum to other solution regions is needed. The path relinking method (Glover 1996), which has been reported to be effective for PFSP-makespan (Nowicki and Smutnicki 2006), could be a good trial for generating a new global best solution that may help RDPSO escape from the local optima.

Lastly, investigations into optimized learning strategies may benefit the performance of RDPSO. This study demonstrates that a proper swarm learning strategy should be a variable strategy that learns more from the personal best solution in the first 500 iterations and then learns more from the global best solution in the second 500 iterations. As mentioned earlier, this idea takes advantage of exploring the solution space in the first 500 iterations and exploiting the solution space in the second 500 iterations. Although this approach is effective, it can be optimized with methods that adjust learning parameters (w, c_1, and c_2) based on the conditions in each iteration. The self-adaptive learning strategy (Wang et al. 2011; Wang and Tang 2012), which adjusts learning parameters in every iteration, has recently received attentions from the researchers of PSO. Although the process of self-adaptive learning strategy is complex, it is a definite future direction for the research of RDPSO.

Acknowledgments This paper was supported in part by the National Science Council, Taiwan, ROC, under the contract NSC101-2221-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

Chen S-H, Chang P-C, Cheng TCE, Zhang Q (2012) A self-guided genetic algorithm for permutation flowshop scheduling problems. Comput Oper Res 39:1450–1457

Dorigo M, Stützle T (2004) Ant colony optimization. MIT, Cambridge

Etiler O, Toklu B, Atak M, Wilson J (2004) A genetic algorithm for flow shop scheduling problems. J Oper Res Soc 55:830–835

Garey MR, Johnson DS, Sethi R (1976) The complexity of flow shop and job shop scheduling. Math Oper Res 1:117–129

Glover F (1996) Tabu search and adaptive memory programming–advances. Applications and challenges. Kluwer, Boston, pp 1–75

Grabowski J, Wodecki M (2004) A very fast tabu search algorithm for the permutation flowshop problem with makespan criterion. Comput Oper Res 31:1891–1909

Jarboui B, Ibrahim S, Siarry P, Abdelwaheb R (2008) A combinatorial particle swarm optimization for solving permutation flowshop problems. Comput Ind Eng 54:526–538

Kennedy J, Eberhart R (1995) Particle swarm optimization Proc AESF Annu Tech Conf 1995 IEEE Int Conf. Neural Netw 4:1942–1948

Kuoa IH, Horng SJ, Kaod TW, Lina TL, Lee CL, 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

Lian Z, Gu X, Jiao B (2008) A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan. Chaos Soliton Fract 35:851–861

Marinakis Y, Marinaki M (2013) Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem. Soft Comput. doi:10.1007/s00500-013-0992-z

Murata T, Ishibuchi H, Tanaka H (1996) Genetic algorithms for flowshop scheduling problems. Comput Ind Eng 30:1061–1071

Nowicki E, Smutnicki C (2006) Some aspects of scatter search in the flow-shop problem. Eur J Oper Res 169:654–666

Nowicki E, Smutnicki C (1996) A fast tabu search algorithm for the permutation flowshop problem. Eur J Oper Res 91:160–175

Ogbu FA, Smith DK (1990) The application of the simulated annealing algorithm to the solution of the n/m/Cmax flow shop problem. Comput Oper Res 17:243–253

Osman I, Potts C (1989) Simulated annealing for permutation flow shop scheduling. OMEGA 17:551–557

Pan Q-K, Tasgetiren MF, Liang Y-C (2008a) A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. Comput Oper Res 35:2807–2839

Pan Q-K, Tasgetiren MF, Liang Y-C (2008b) A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Comput Ind Eng 55:795–816

Pan Q-K, Tasgetiren MF, Suganthan PN, Chua T-J (2011) A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Inf Sci 12:2455–2468

Ponnambalm SG, Jawahar N, Chandrasekaran S (2009) Discrete particle swarm optimization algorithm for flowshop scheduling. In: Lazinica A (ed) Particle swarm optimization. InTech, Vienna

Rajendran C, Ziegler H (2004) Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. Eur J Oper Res 155:426–438

Rameshkumar K, Suresh RK, Mohanasundaram KM (2005) Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan. Lect Notes Comput Sci 3612:572–581

Reeves CR (1995) A genetic algorithm for flow shop sequencing. Comput Oper Res 22:5–13

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

Sipper D, Bulfin R (1997) Production: planning, control, and integration. The McGraw-Hill, New York
Taillard E (1990) Some efficient heuristic methods for the flow shop sequencing problem. Eur J Oper Res 47:65–74
Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64:278–285
Tasgetiren MF, 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. Eur J Oper Res 177:1930–1947
Wang X, Tang L (2012) A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking. Appl Soft Comput 12:652–662
Wang Y, Li B, Weise T, Wang J, Yuan B, Tian Q (2011) Self-adaptive learning based particle swarm optimization. Inf Sci 181:4515–4538
Ying K-C, Liao C-J (2004) An ant colony system for permutation flowshop sequencing. Comput Oper Res 31:791–801
Zhang C, Jiaxu N, Dantong O (2010a) A hybrid alternate two phases particle swarm optimization algorithm for flow shop scheduling problem. Comput Ind Eng 58:1–11
Zhang J, Zhang C, Liang S (2010b) The circular discrete particle swarm optimization algorithm for flow shop scheduling problem. Expert Syst Appl 37:5827–5834
Zhigang L, Xingsheng G, Bin J (2006) A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. Appl Math Comput 175:773–785
Zobolas GI, Tarantilis CD, Ioannou G (2009) Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. Comput Oper Res 36:1249–1267