Artif Life Robotics (2016) 21:18–23
DOI 10.1007/s10015-015-0259-0

ORIGINAL ARTICLE

A new algorithm for flexible job-shop scheduling problem based on particle swarm optimization

Wannaporn Teekeng[1] · Arit Thammano[2] · Pornkid Unkaw[3] · Jiraporn Kiatwuthiamorn[2]

Received: 7 February 2015 / Accepted: 7 December 2015 / Published online: 18 January 2016
© ISAROB 2016

Abstract This paper proposes a new algorithm, named EPSO, for solving flexible job-shop scheduling problem (FJSP) based on particle swarm optimization (PSO). EPSO includes two sets of features for expanding the solution space of FJSP and avoiding premature convergence to local optimum. These two sets are as follows: (I) particle life cycle that consists of four features: (1) courting call—increasing the number of more effective offspring (new solutions), (2) egg-laying stimulation—increasing the number of offspring from the better parents (current solutions), (3) biparental reproduction—increasing the diversity of the next generation (iteration) of solutions, and (4) population turnover—succeeding the population (the current set of all solutions) in the previous generation by a population in a new generation that is as able but more diverse than the previous one; and (II) discrete position update mechanism—moving particles (solutions) towards the flight leader (the best solution), namely, interchanging some integers in every solution with those in both the best solution and itself, using similar swarming strategy as the update procedure of the continuous PSO. The basic objective function used was to minimize makespan which is the most important objective, hence, providing the simplest way to measure the effectiveness of the generated solutions. Benchmarking EPSO with 20 well-known benchmark instances against two widely-reported optimization methods demonstrated that it performed either equally well or better than the other two.

Keywords Particle swarm optimization · Flexible job-shop scheduling · Meta-heuristic · Life cycle

1 Introduction

Flexible job-shop scheduling problem (FJSP) is a type of optimization problem. In this context, scheduling is the allocation and arrangement of resources over time to perform a collection of tasks to achieve an objective or goal. Scheduling varies according to the constraints of different conditions or situations. Normally, scheduling is graphically represented by Gantt chart that shows resource allocation and scheduling arrangement where the *y*-axis represents a variety of resources and the *x*-axis represents the length of time that each resource is utilized. FJSP is an extension of the classical job-shop scheduling problem (JSSP), but is much harder and more complex to solve because FJSP allows each operation to be processed by more than one machine and each machine can finish each operation in a different amount of time.

Recently, a number of meta-heuristic approaches, such as genetic algorithm (GA) [1], ant colony optimization (ACO) [2], shuffled frog leaping algorithm (SFLA) [3], particle swarm optimization (PSO) [4], artificial immune algorithm (AIA) [5], and harmony search (HM) [6], have gained a lot of attention from researchers as viable FJSP optimization methods. Pezzella et al. [7] proposed a genetic

This work was presented in part at the 20th International Symposium on Artificial Life and Robotics, Beppu, Oita, January 21–23, 2015.

✉ Wannaporn Teekeng
w_teekeng@hotmail.com; wannaporn@rmutl.ac.th

[1] Rajamangala University of Technology Lanna, Tak, Thailand

[2] King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand

[3] Rajamangala University of Technology Phra Nakhon, Bangkok, Thailand

algorithm that incorporates different strategies for generating initial population and selecting individuals for reproduction. Zhang et al. [8] proposed new GA concepts for generating high-quality initial population, called global selection (GS) and local selection (LS). In another study, Zhang et al. [9] used variable neighborhood search to perform local search in PSO. Bagheri et al. [10] used an artificial immune algorithm to solve FJSP. They proposed a method for constructing diverse initial population, a strategy of using 'most work remaining' and 'most operation remaining' to arrange the order of operations, and a mutation procedure to achieve even more diversity. Teekeng et al. [11] proposed an SFLA-FS algorithm that incorporates fuzzy logic for selecting parents that are better than those in the previous generation. In another one of their studies, Teekeng et al. [12] introduced new crossover and mutation features into GA for solving FJSP. Yuan et al. [13] presented a hybrid harmony search (HM) for solving FJSP.

Our proposed EPSO algorithm introduced the following two sets of new features to the original concept of PSO: (I) particle life cycle that consists of 4 features: (1) courting call, (2) egg-laying stimulation, (3) biparental reproduction, and (4) population turnover; and (II) discrete position update mechanism.

This paper is organized as follows: Sect. 2 briefly describes FJSP; Sect. 3 describes the original PSO; Sect. 4 presents our proposed EPSO algorithm for FJSP; Sect. 5 presents the performance test results; and Sect. 6 concludes the paper.

2 FJSP

Flexible job-shop scheduling problem (FJSP) allows a job operation to be processed by any machine out of a set of several machines. To solve FJSP is to find the best schedule for a set of R jobs $J = \{J_1, J_2, \ldots, J_R\}$ that is operated by a set of S machines $M = \{M_1, M_2, \ldots, M_S\}$. Each job can have a different set of operations, and each operation O_{ij}, the jth operation of the ith job, can be processed by any of the available machines. An example of FJSP is given in Table 1. Each row refers to an operation and each column refers to a machine. For example, the first row shows that the first operation of Job1 can be processed by M_1 and M_2 using 2 and 4 time units, respectively. On the other hand, the fifth row shows that the third operation of the second job is allowed to be processed only by M_1.

Table 1 An example of FJSP with 4 jobs, 2 machines, and 9 operations

Job	Operation	Machine processing time	
		M_1	M_2
J_1	O_{11}	2	4
	O_{12}	5	4
J_2	O_{21}	1	4
	O_{22}	6	5
	O_{23}	2	–
J_3	O_{31}	4	6
	O_{32}	–	5
J_4	O_{41}	3	4
	O_{42}	5	7

3 Particle swarm optimization algorithm

Particle swarm optimization (PSO) is an algorithm that was inspired by the behaviour of swarming animals like a flock of birds or a school of fish [4]. It is similar to genetic algorithm (GA) in that they both randomly select an initial population and improve it so that the population of the next generation (iteration) is better at finding the optimal solution (the actual bound) than the previous one. In PSO algorithm, a potential solution is called a particle. The ultimate goal of a particle is to reach the optimal solution. A particle moves towards the solution by positioning itself nearer to the flight leader, Gbest, that is the nearest particle to the optimal solution in a particular iteration. The position of a particle is updated according to Eqs. (1) and (2):

$$v'_{\mathrm{id}} = \omega \times v_{\mathrm{id}} + c_1 \times \mathrm{Rand} \times (p_{\mathrm{id}}^{\mathrm{best}} - p_{\mathrm{id}}) + c_2 \times \mathrm{Rand} \times (p_{\mathrm{gd}}^{\mathrm{best}} - p_{\mathrm{id}}) \quad (1)$$

$$p_{\mathrm{id}} = p_{\mathrm{id}} + v'_{\mathrm{id}} \quad (2)$$

where v'_{id} is the speed of the id$^{\mathrm{th}}$ particle in the current iteration; v_{id} is the speed of the id$^{\mathrm{th}}$ particle in the previous iteration; ω is the initial weight; c_1 and c_2 are positive constants; Rand is a random function in the range of [0, 1]; $p_{\mathrm{id}}^{\mathrm{best}}$ is the best local position of the id$^{\mathrm{th}}$ particle (Pbest); $p_{\mathrm{gd}}^{\mathrm{best}}$ is the best global position among all particles (Gbest); and p_{id} is the id$^{\mathrm{th}}$ particle. PSO is widely used for a variety of optimization tasks; however, it cannot be applied directly to FJSP because PSO was developed to solve continuous optimization problems but FJSP is a combinatorial problem which is discrete in nature.

4 EPSO algorithm for FJSP

Our EPSO algorithm uses the same position update strategy as PSO does but defines a new discrete particle representation and a new position update mechanism that suits

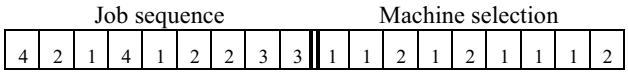


Fig. 1 A particle representation

the discrete nature of FJSP. In this section, we describe this representation and mechanism, our objective function, and particle life cycle, a set of enhancing features.

4.1 Particle representation

For discrete FJSP problem, a particle is defined as follows. Each particle consists of a string of integers separated into two parts: (1) job sequence part containing as many slots, each holding a single job number, as the total number of operations, and (2) machine selection part containing the same number of slots as that of the job sequence part, each holding a single machine number assigned for a job operation of that job. The first integer slot (or dimension) of the job sequence part denotes that the operation of the selected job in this slot is performed first and the corresponding slot of the machine selection part denotes the machine selected to perform that operation; similarly, the second dimension denotes that the second operation (of any one of the jobs selected) is performed by the machine selected to perform it, and so on. To make particle representation clear, we show a concrete example in Fig. 1 and Table 1. This example is a string of 18 dimensions (9×2) representing a particle. Initially, the job sequence part contains 9 randomly selected job numbers, O_{ij}, $1 \leq j \leq n_i$, where n_i is the number of operations, while the machine selection part contains 9 methodically selected feasible machine numbers, M_{ij}. The machine selection method is as follows: (1) 80 % of the population consist of machines that perform the operations of jobs in the shortest processing time (SPT); and (2) 20 % of the population consist of randomly selected machines that can perform the selected jobs. The processing time of each machine is P_{ijk}, $1 \leq k \leq M_{ij}$. It can be seen in Fig. 1 that the first slot operation is the first job operation of job 4 with machine 1; the second slot operation is the first job operation of job 2 with machine 1; while the last slot operation is the second job operation of job 3 with machine 2.

4.2 Objective function

After a particle is represented, the particle is measured for its search effectiveness by an objective function. The objective function used in this study minimizes the maximums of the complete-time of every job (i.e., minimizes makespan),

$$C\max = \max_{1 \leq j \leq n}\{C_j\} \tag{3}$$

where C_j is the complete time of job j.

This objective function is also used for measuring the effectiveness of particles when their positions at the end of an iteration are updated and when the particle life cycle features generate new positions.

4.3 Particle life cycle

Enhancing this discrete algorithm based on PSO by incorporating the following four particle life cycle features brings about changes to particle population as described below.

The first feature is courting call ($p_{\mathrm{id}}^{\mathrm{call}}$). The courting call of a particle (solution) is an indicator of the effectiveness of the particle; the more effective the particle, the louder the call, and the higher the chance that the particle will take a mate and reproduce, hence, there will be more effective and diverse offspring from the current generation. A courting call is a measure of the extent to which every machine can perform every of its assigned operation in roughly the same amount of time. The loudest call means that the total processing time of all operations processed by that machine of that particle is close to the average processing time of all operations, as expressed in Eq. 4 below,

$$p_{\mathrm{id}}^{\mathrm{call}} = \sum_{k=1}^{n} |m_k - \overline{\mathrm{mac}}_{\mathrm{id}}| \tag{4}$$

Figure 2 illustrates $p_{\mathrm{id}}^{\mathrm{call}}$ of two particles—p_1^{call} and p_2^{call}—of which p_1^{call} is louder than p_2^{call} because the first particle operates all of its machines in an amount of time closer to $\overline{\mathrm{mac}}_{\mathrm{id}}$ than the second particle does: $\overline{\mathrm{mac}}_1$ of p_1^{call} is 16.5 while its m_1 and m_2 are 16 and 17, respectively, while $\overline{\mathrm{mac}}_2$ of p_2^{call} is 19 which is farther to its m_1 and m_2 at 12 and 26, respectively.

The second feature is egg-laying stimulation ($q_{\mathrm{id}}^{\mathrm{ovum}}$). $q_{\mathrm{id}}^{\mathrm{ovum}}$ is the variable number of offspring that can be produced by a courting call of particle. A courting call without egg-laying stimulation produces a fixed number of eggs. With egg-laying stimulation, a courting call produces a variable number of eggs that depends on the effectiveness of the particle. Because of this arrangement, the next

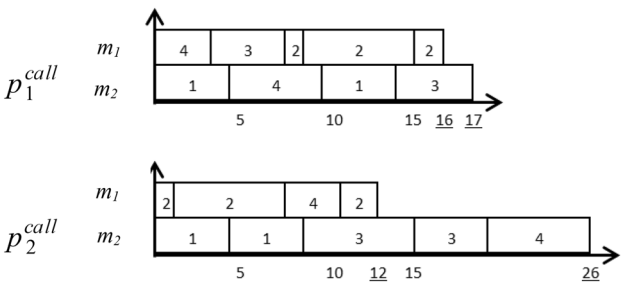


Fig. 2 Gantt chart of courting calls of two particles

generation (iteration) will have a higher number of more effective particles. Mathematically, $q_{\mathrm{id}}^{\mathrm{ovum}}$ is expressed by the equation below,

$$q_{\mathrm{id}}^{\mathrm{ovum}} = \max\left\{E_{\min}, E_{\max} - \left[\mathrm{round}\left(\frac{E_{\max} \times p_{\mathrm{id}}^{\mathrm{call}}}{\max\left(P_{\mathrm{id}}^{\mathrm{call}}\right)}\right)\right]\right\} \tag{5}$$

where $E_{\min}$ is the minimum number of eggs a particle lays and $E_{\max}$ is the maximum.

The third feature is biparental reproduction. This kind of reproduction exchanges job numbers in the job sequence part of two parent particles. This exchange increases the diversity of offspring, leading to a better chance for the search to avoid local optimums. In this study, only the job sequence part is crossed over, not the machine selection part. The steps in the crossover procedure, illustrated in Fig. 3, are as follows:

(a) Randomly select half of the jobs from a randomly selected elite parent—the best 30 % of all of the particles;
(b) Copy all of the selected jobs from the elite parent and place them in the same slot of the offspring;
(c) Fill in the rest of the slots sequentially from left to right with the jobs from the non-elite parent (the last 70 % of particles) that are different from those jobs in b.
(d) Repeat step a–c with the same parents until the maximum number of eggs allowed is reached.

The last feature is population turnover (p_{ini}). p_{ini} selects the best 80 % of the initial population and adds newly randomly selected particles (p_{new}) to make up 100 %. Therefore, most of the best particles in the current generation (iteration) will be passed along to the next generation while significant diversity is introduced. The equation for p_{ini} is below,

$$p_{\mathrm{ini}} = (p_{\mathrm{ini}} \times 0.8) + p_{\mathrm{new}} \tag{6}$$

4.4 Discrete particle position update mechanism

Similar to PSO, EPSO has a position update mechanism to move particles closer to Gbest, the flight leader, the one

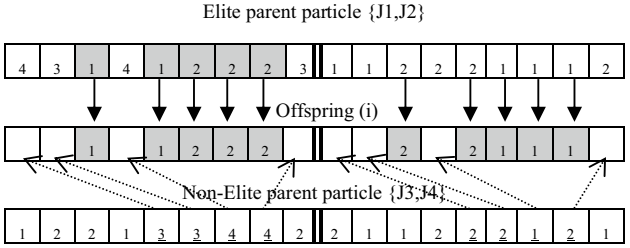
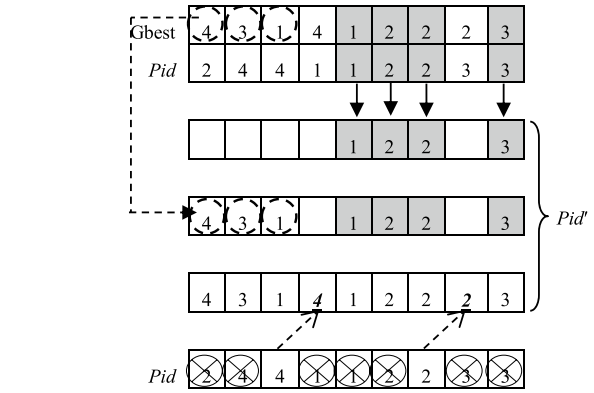
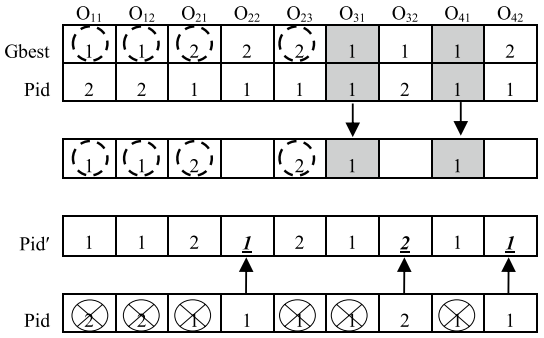


Fig. 3 Biparental reproduction



(a) Updating job sequence part with Gbest



(b) Updating machine selection part with Gbest

Fig. 4 An example of particle position update mechanism

EPSO Algorithm

Step 1: Set parameter values: population size, termination criteria, number of elites, number of eggs, number of matings an elite does
Step 2: Initialize particles (job sequence part and machine selection part)
Step 3: Evaluate the effectiveness of each particle in the population by the objective function
Step 4: If the termination criterion (80% of the population share the same effective values) is not met, go to step 5; otherwise, go to step 9
Step 5: Perform courting call, egg-laying stimulation, and biparental reproduction (particle life cycle)
Step 6: Update position of every particle
Step 7: Update Gbest and Pbest for the next iteration
Step 8: Turn the population over (particle life cycle), then go to step 4
Step 9: Show result

Fig. 5 Pseudo-code of EPSO algorithm for discrete FJSP problem. The *italicized* texts represent new features to the original PSO

nearest to the solution, thus increasing the chance of a particle to reach the optimal solution. PSO cannot be applied directly to FJSP because its update mechanism uses real numbers; instead, EPSO uses discrete integers for the same purpose. To take advantage of the combined effectiveness of the flight leader and itself, the next position of a particle depends on both Gbest and Pbest values of the flight leader and itself, respectively. When these two values are dissimilar, the particle will be positioned somewhere in the middle between them. An example showing the discrete update mechanism is depicted in Fig. 5. This example shows changes in job sequence part and machine selection part of a particle as it is updated with Gbest. The same procedure applies when the particle is further updated with Pbest. The update procedure is as follows:

- *Updating job sequence part*: the job numbers in the corresponding slots of Gbest and a pre-updated particle are compared, and the slots holding similar job numbers in the pre-updated particle are kept unchanged while the slots holding different job numbers are changed; Fig. 4a shows an example of these changes; the job numbers in the 4 gray slots are similar and so remain the same, while the other 5 slots are dissimilar and the following changes are made: randomly select the job number and job operation of a half of the dissimilar slots of Gbest and sequentially place them in the similar slot of the pre-updated particle; then, the other half of the dissimilar slots of the pre-updated particle are sequentially filled with the job numbers that are required to pair with all of the job operations that are not selected in the former step, as shown Fig. 4a.
- *Updating machine selection part*: As shown in Fig. 4b, after a rearrangement that is reversed after the update, the machine numbers in the corresponding slots of Gbest and the pre-updated particle are compared, and the slots holding similar machine numbers in the pre-updated particle are kept unchanged while the slots holding different job numbers are changed; the changes are made in the same way as those made in the job sequence part.

To summarize the steps in our proposed algorithm, we present them in pseudocode in Fig. 5 below.

Table 2 Best solutions and percentage relative errors (in parentheses) from the lower bounds of Fattahi's data set

Test Instance	Job and machine ($n \times m$)	Lower bound	Bagheri et al. [10] AIA method	Demir and Isleyen [15] Mathematical model	EPSO
SFJS1	2 × 2	66	66 (0.00)	66 (0.00)	66 (0.00)
SFJS2	2 × 2	107	107 (0.00)	107 (0.00)	107 (0.00)
SFJS3	3 × 2	221	221 (0.00)	221 (0.00)	221 (0.00)
SFJS4	3 × 2	355	355 (0.00)	355 (0.00)	355 (0.00)
SFJS5	3 × 2	119	119 (0.00)	119 (0.00)	119 (0.00)
SFJS6	3 × 3	320	320 (0.00)	320 (0.00)	320 (0.00)
SFJS7	3 × 5	397	397 (0.00)	397 (0.00)	397 (0.00)
SFJS8	3 × 4	253	253 (0.00)	253 (0.00)	253 (0.00)
SFJS9	3 × 3	210	210 (0.00)	210 (0.00)	210 (0.00)
SFJS10	4 × 5	516	516 (0.00)	516 (0.00)	516 (0.00)
MFJS1	5 × 6	396	468 (18.18)	468 (18.18)	468 (18.18)
MFJS2	5 × 7	396	448 (13.13)	446 (12.63)	446 (12.63)
MFJS3	6 × 7	396	468 (18.18)	466 (17.68)	466 (17.68)
MFJS4	7 × 7	496	554 (11.69)	564 (13.71)	554 (11.69)
MFJS5	7 × 7	414	527 (27.29)	514 (24.15)	514 (24.15)
MFJS6	8 × 7	469	635 (35.39)	634 (35.18)	634 (35.18)
MFJS7	8 × 7	619	879 (42.00)	928 (49.92)	879 (42.00)
MFJS8	9 × 8	619	884 (42.81)	–	884 (42.81)
MFJS9	11 × 8	764	1088 (42.41)	–	1059 (38.61)
MFJS10	12 × 8	944	1267 (34.22)	–	1205 (27.65)
Average relative error			14.27	–	13.53
			–	10.11***	9.50***

The symbol "–" means that the machine cannot execute the corresponding operation

5 Results of performance test

Our proposed algorithm was performance tested with Fdata set, a benchmark data set invented by Fattahi [14]. An Fdata set consists of 20 test instances that are grouped according to their size: (1) small size (SJJS1:10), and (2) medium and large size (MFJS1:10). It is relatively easy to find a solution that matches the lower bound of SJJS1:10, but it is not so for MFJS1:10.

The test parameters used were as follows:

- Population size: initially 200 and does not increase over 500.
- Termination check: 80 % of the population share the same effectiveness values.
- Number of elite particles: the best 30 % of the population.
- Number of eggs: $E_{\max} = 5$ to $E_{\min} = 2$.
- Number of matings (for elite parent particle) $= 3$.

Performance was measured in terms of closeness to the lower bound of the benchmark. The best solution that the algorithm found was reported, and so was the average percentage of relative error from the lower bound of the final solution. The percentage of relative error RE (%) is calculated by the following Eq. (7):

$$\mathrm{RE}(\%) = \frac{C_{\mathrm{best}} - \mathrm{BKS}}{\mathrm{BKS}} \times 100 \tag{7}$$

where C_{best} is the best solution obtained from our algorithm.

BKS is the best-known solution or the lower bound of the benchmark.

Table 2 shows the best results from our algorithm for all 20 test instances of the Fdata set (SFJS1:10 and MFJS1:10) compared to the best results obtained from the artificial immune algorithm (AIA) of Bagheri et al. [8] as well as our best results for 17 instances of the Fdata set (SFJS1:10 and MFJS1:7) compared to the best results obtained from a mathematical model proposed by Demir et al. [15]. With respect to the results obtained from testing with SFJS1:10, the best solutions from our algorithm, AIA algorithm, and Demir's mathematical model were the same and equal to the lower bound, but with respect to the results obtained from testing with MFJS, the best solution from our algorithm was either the same or better than both the best solutions from AIA and Demir's mathematical model. The average percentages of relative error from the lower bound of our algorithm and the AIA algorithm for the 20 test instances were 13.53 and 14.27 %, respectively, and the average percentages of relative error from the lower bound of our algorithm and Demir's mathematical model for the 17 test instances were 9.50*** and 10.11 %***, respectively. All of these results show that the performance of EPSO was better than those of AIA and Demir's mathematical model.

6 Conclusion

This paper proposes an algorithm for solving discrete flexible job-shop scheduling problem (FJSP) based on the swarming strategy of the particle swarm optimization (PSO) algorithm. Two sets of enhancing features are introduced: (I) particle life cycle that consists of the following features: (1) courting call, (2) egg-laying stimulation, (3) biparental reproduction, and (4) population turnover; and (II) discrete position update mechanism. The performance test results show that the proposed algorithm performed better than AIA algorithm and Demir's mathematical model. In our future work, we will attempt to apply this algorithm to a much more complex FJSP with multi-objective functions.

References

1. Goldberg DE (1989) Genetic algorithms in search optimisation and machine learning. Addison-Wesley, Reading
2. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization: artificial ant as a computational intelligence technique. IRIDIA Technical Report Series, University Libre De Bruxelles, Belgium
3. Eusuff MM, Lansey KE (2003) Optimization of water distribution network design using the shuffled frog leaping algorithm. J Water Resour Plan Manage 129(3):210–225
4. Kennedy J, Eberhard R (1995) Particle swarm optimization. Phys Rev B 13:5344–5348
5. Dasgupta D (2002) Special issue on artificial immune system. IEEE Trans Evol Comput 6:225–256
6. Geem ZW, Kim JH, Loganathan GV (2001) A new heuristic optimisation algorithm: harmony search. Simulation 76:60–68
7. Pezzella F, Morganti G, Ciaschetti G (2008) A genetic algorithm for the flexible job-shop scheduling problem. Comput Oper Res 35(10):3202–3212
8. Zhang GH, Gao L, Shi Y (2011) An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Syst Appl 38(4):3563–3573
9. Zhang G, Gao L, Li X (2013) Solving the flexible job-shop scheduling problem using particle swarm optimization and variable neighborhood search. Int J Adv Comput Technol 5(4):291–299
10. Bagheri A, Zandieh M, Mahdavia I, Yazdani M (2010) An artificial immune algorithm for the flexible job-shop scheduling problem. Future Gener Comput Syst 26:533–541
11. Teekeng W, Thammano A (2011) A combination of Shuffled frog leaping algorithm and fuzzy logic for flexible job-shop scheduling problems. Proc Comput Sci Complex Adapt Syst 6:69–75
12. Teekeng W, Thammano A (2012) Modified genetic algorithm for flexible job-shop scheduling problems. Proc Comput Sci Complex Adapt Syst 12:122–128
13. Yuan Y, Xu H, Yang J (2013) A hybrid harmony search algorithm for the flexible job shop scheduling problem. Appl Soft Comput 13(7):3259–3272
14. Fattahi P, Mehrabad MS, Jolai F (2007) Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. J Intell Manuf 18(3):331–342
15. Demir Y, Isleyen SK (2013) Evaluation of mathematical models for flexible job-shop scheduling problems. Appl Math Model 37(3):977–988