Int J Adv Manuf Technol (2012) 62:267–277
DOI 10.1007/s00170-011-3807-3

ORIGINAL ARTICLE

A swarm optimization approach for flexible flow shop scheduling with multiprocessor tasks

Manas Ranjan Singh · S. S. Mahapatra

Received: 1 September 2011 / Accepted: 21 November 2011 / Published online: 4 December 2011

Abstract In simple flow shop problems, each machine operation center includes just one machine. If at least one machine center includes more than one machine, the scheduling problem becomes a flexible flow shop problem (FFSP). Flexible flow shops are thus generalization of simple flow shops. Flexible flow shop scheduling problems have a special structure combining some elements of both the flow shop and the parallel machine scheduling problems. FFSP can be stated as finding a schedule for a general task graph to execute on a multiprocessor system so that the schedule length can be minimized. FFSP is known to be NP-hard. In this study, we present a particle swarm optimization (PSO) algorithm to solve FFSP. PSO is an effective algorithm which gives quality solutions in a reasonable computational time and consists of less numbers parameters as compared to the other evolutionary metaheuristics. Mutation, a commonly used operator in genetic algorithm, has been introduced in PSO so that trapping of solutions at local minima or premature convergence can be avoided. Logistic mapping is used to generate chaotic numbers in this paper. Use of chaotic numbers makes the algorithm converge fast towards near-optimal solution and hence reduce computational efforts further. The performance of schedules is evaluated in terms of total completion time or makespan (C_{max}). The results are presented in terms of percentage deviation (PD) of the solution from the lower bound. The results are compared with different versions of genetic algorithm (GA) used for the purpose from open literature. The results indicate that the proposed PSO algorithm is quite effective in reducing makespan because average PD is observed as 2.961, whereas GA results in average percentage deviation of 3.559. Finally, influence of various PSO parameters on solution quality has been investigated.

Keywords Flexible flow shop · PSO · Mutation · Chaotic numbers · Makespan

Electronic supplementary material The online version of this article (doi:10.1007/s00170-011-3807-3) contains supplementary material, which is available to authorized users.

M. R. Singh · S. S. Mahapatra (✉)
Department of Mechanical Engineering,
National Institute of Technology,
Rourkela, India
e-mail: mahapatrass2003@yahoo.com

M. R. Singh
e-mail: manasranjan.singh@gmail.com

1 Introduction

Scheduling is basically concerned with allocation of operations on machines (i.e., a sequence of operations on machines) in such a manner that some performance goals such as flow time, tardiness, lateness, and makespan can be met. The scheduling procedures developed so far are quite realistic, fast solution generating capability and find widespread applications in industries. Basically, job scheduling is of two types—job shop and flow shop scheduling. A classic job shop scheduling problem (JSP) is a set of n jobs processed by a set of m machines. Each job is processed on machines in a given order with a given processing time and each machine can process only one. The flexible job shop scheduling problem (FJSP) is an extension of the classical job shop problem where operations are allowed to be processed on any among a set of available machines. FJSP is more difficult than the classical JSP because it contains an additional constraint of assigning operations to machines.

The flow shop problem consists of two main elements, i.e., a group of m machines and a set of n jobs to be processed on this group of machines. Flow shop is a typical combinatorial optimization problem where each job has to go through the processing in each and every machine on the shop floor. Each machine has same sequence of jobs. The jobs have different processing time for different machines. A flexible flow shop problem (FFSP) is an extended development of the classical flow shop, which consists of two or more production stages in series and having one or more parallel machines at each stage. Scheduling jobs in flexible flow shops is considered as NP-hard problem. For such problems, it is not always possible to find an optimal solution in a reasonable time. It has been proved that a two-stage flexible flow shop scheduling problem is NP-hard in the strong sense even if there is only one machine on the first stage and two machines on the second stage [1, 2]. Hence, a variety of heuristic procedures such as dispatching rules, local search, and metaheuristic procedures like tabu search (TS), simulated annealing (SA), particle swarm optimization (PSO), and genetic algorithm (GA) are used to solve such problems and to generate approximate solutions close to the optimum with considerably less computational time. Recently, multistage flexible flow shop scheduling problem has received attention due to its theoretical and practical significance. The FFSP scheduling has been widely applied in different manufacturing environments like iron and steel, textile, and electronic industries [3]. The first model on FFSP has been proposed by Arthanari and Ramamurthy using branch and bound algorithm [4]. Brah and Hunsucker [5] and Moursli and Pochet [6] have proposed a branch and bound algorithm to solve the flexible flow shop problems. To obtain a near-optimal solution, metaheuristics have also been proposed. Oguz and Ercan [7] have proposed a genetic algorithm for the hybrid flow shop scheduling with multiprocessor task problems using a new crossover operator. They performed a preliminary test to set the best values of the control parameters, viz., population size, crossover rate, and mutation rate.

Particle swarm optimization, first proposed by Kennedy and Eberhart [8], is one of the most recent and hopeful evolutionary metaheuristics, which is inspired by adaptation of a natural system based on the metaphor of social communication and interaction. Originally, PSO was focused on solving nonlinear programming and nonlinear constrained optimization problems comprising of continuous variables. Later, the algorithm is applied to various scheduling problems with improved performance. Xia et al. [9] have demonstrated the application of PSO to well-known job shop scheduling problem. PSO is an effective algorithm which gives high-quality solutions in a reasonable computational time and consists of less numbers parameters as compared to the other evolutionary metaheuristics like GA [10]. In this paper, the search mechanism of the particle swarm optimization is considered due its effectiveness. An effective solution approach is proposed for solving the flexible flow shop scheduling problem. The proposed approach uses PSO to assign jobs on machines at each stage and schedule job sequence on each machine in the corresponding stages. The objective which is considered in this paper is to minimize makespan. However, PSO has an inherent drawback. It gets trapped at local optimum due to large reduction in velocity values as iteration proceeds and hence sometimes do not reach at optimal solution. This drawback has been eliminated effectively by incorporating mutation, a commonly used operator in genetic algorithm, to improve the solution quality. In this paper, chaotic numbers are generated using logistic mapping instead of random numbers.

2 Literature review

In the past, scheduling problem in a hybrid or flexible flow shop has received attention of researchers because of its importance from both theoretical and practical points of view. This problem has been showed as an adequate model for the study of a great number of production systems, specifically process industries such as glass, paper, metallurgy, wood, textile, and aerospace. Jayamohan and Rajendran [11] have investigated the effectiveness of two approaches using different dynamic dispatching rules for the scheduling of flexible flow shops minimizing the flow time and tardiness of the jobs. One approach is to use of different dispatching rules at different stages of the flow shop and the other suggesting the use of the same dispatching rule at all the stages of the flow shop. Kurz and Askin [12] have presented a hybrid flexible flow line environment with identical parallel machines and non-anticipatory sequence-dependent setup times with an objective to minimize the makespan. Bertel and Billaut [13] have investigated an industrial application based on a three-stage hybrid flow shop scheduling problem with recirculation. The problem is to perform jobs between a release date and a due date in order to minimize the weighted number of tardy jobs. An integer linear programming formulation was proposed to solve the problem. Xiao et al. [14] have used GA to search the permutation space for the solution of the m-stage problem with makespan as objective. The same problem, with the addition of sequence-dependent setup times, was approached by Kurz and Askin [15] using GA with the random key representation proposed by Bean [16]. Ruiz and Maroto [17] have employed GA to minimize makespan in a K-stage problem with unrelated parallel machines, sequence-dependent setup times, and machine reliability.

3 Particle swarm optimization

Particle swarm optimization, inspired by social behavior of bird flocking and fish schooling, is a population-based, bio-inspired optimization algorithm, originally introduced by Kennedy and Eberhart [8]. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. Each member in PSO is called particle. These particles are moved around in the search-space according to a few simple formulae. The movements of the particles are guided by their own best known position in the search space as well as the entire swarm's best known position. Generally, PSO is characterized as a simple heuristic of well-balanced mechanism with flexibility to enhance and adapt to both global and local exploration abilities. Compared with GA, PSO has some attractive characteristics. It has memory that enables to retain knowledge of good solutions by all particles whereas previous knowledge of the problem is destroyed once the population changes in GAs. In GAs, chromosomes share information with each other. So the whole population moves like a one group towards an optimal area. In PSO, only global best gives out the information to others. It is a one-way information sharing mechanism. The evolution only looks for the best solution. Compared with GA, all the particles tend to converge to the best solution quickly even in the local version in most cases. PSO does not require that the optimization problem be differentiable as is required by classical optimization methods such as gradient descent and quasi-Newton methods. PSO can therefore also be used on optimization problems that are partially irregular and noisy. Due to the simple concept, easy implementation, and quick convergence, PSO has gained much attention and been successfully applied to a wide range of applications such as power and voltage control, neural network training, mass–spring system, task assignment, supplier selection and ordering problem, automated drilling, state estimation for electric power distribution systems, etc. [18–23]. In this work, the required parameters are initialized and the initial population is generated using chaotic numbers. After evaluation, the PSO algorithm repeats the following steps iteratively:

- Personal best (best value of each individual so far) is updated if a better value is discovered.
- Then, the velocities of all the particles are updated based on the experiences of personal best and the global best in order to update the position of each particle with the velocities currently updated.
- Permutation is determined through an encoding scheme so that evaluation is again performed to compute the fitness of the particles in the swarm.

After finding the personal best and global best values, velocities and positions of each particle are updated using Eqs. 1 and 2, respectively.

$$v_{ij}^{t} = w^{t-1} v_{ij}^{t-1} + c_1 r_1 \left(p_{ij}^{t-1} - x_{ij}^{t-1}\right) + c_2 r_2 \left(g_{j}^{t-1} - x_{ij}^{t-1}\right) \tag{1}$$

$$x_{ij}^{t} = x_{ij}^{t-1} + v_{ij}^{t} \tag{2}$$

where v_{ij}^{t} represents velocity of particle i at iteration t with respect to jth dimension (j=1,2,……n). p_{ij}^{t} represents the position value of the ith personal best with respect to the jth dimension. x_{ij}^{t} is the position value of the ith particle with respect to jth dimension. c_1 and c_2 are positive acceleration parameters, called cognitive and social parameter, respectively, and r_1 and r_2 are chaotic numbers between (0,1). w is known as inertia weight, which is updated as:

$$w^{t} = w^{t-1} \times \alpha \tag{3}$$

where α is a decrement factor. The higher the value of w, the higher is the impact of the previous velocities on the current velocity. The algorithm terminates after repeating for a maximum number of iterations.

Conventionally, PSO was used for solving unconstrained uni-objective optimization problem only. Due to its simplicity and efficiency, numerous efforts were made to extend PSO in various aspects. Now, multi-objective optimization is involved in most real-world problems. An introduction of PSO to multi-objective optimization problem (MOP) was naturally encouraged. In 2002, one of the first PSO applications to MOP was introduced by Hu and Eberhart [24], where PSO was modified by using a dynamic neighborhood strategy, new particle memory updating, and one-dimension optimization to deal with two-objective MOPs. Each generation begins with each particle calculating its distance to every other particle in term of the fitness given by the first objective. Next, the two closest neighbors are located and the local best is selected from the best neighbor base on the second objective. The best previous solution, Pbest, is maintained for each particle and will be replaced if and only if a better solution that dominates the previous one is found. Mostaghim and Teich [25] have proposed the sigma method for selecting the best local guides. Here, sigma values are calculated and assigned to the particles in the archive as well as those living in the swarm. Then, for each particle in the swarm, a particle in the archive with the closest sigma value is selected as its guide. Wang and Singh [26] have proposed constrained multi-objective particle swarm optimization where the non-dominated solutions are stored in an archive and the gbest is selected from the archive with binary tournament procedure. A rejecting strategy is incorporated to check the constraint violation of the candidate solutions.

The solution is considered feasible and be allowed to be compared with the non-dominated solutions stored in the archive if and only if all the constraints are satisfied. Else, the solution is regarded as infeasible.

3.1 Chaotic number

Chaotic numbers are numbers generated by using mathematical formula. In this paper, chaotic numbers are generated using logistic mapping. Chaotic numbers are deterministic but unpredictable due to their high sensitivity to initial conditions. They, though periodic in nature, have very long periods hence can be considered as non-periodic for computation purpose. They also exhibit uniform distribution nature and have high entropy (degree of disorder). The reason behind opting chaotic sequences is due to their ability to converge fast toward optimal solution, while retaining a proper balance between exploitation and exploration. From the above discussion, it is clear that chaotic numbers can be efficiently used as random number generator. Chaotic numbers have been applied in various fields, such as secure transmission, natural modeling phenomena, etc., and obtained very interesting results.

In nearly every kind of probabilistic distribution and to generate randomness, random number generators (RNGs) have been widely used in algorithms. RNGs are known for slow convergence and have an inherent characteristic of sticking to a solution. To overcome this difficulty, chaotic generators have recently been used instead of RNGs. The following equation is used to replace RNGs [27]:

$$N(\mathrm{t}) = R \times N(t-1) \times (1 - N(t-1)) \qquad (4)$$

The logistic map illustrated in Eq. 4 [28] is one of the simplest dynamic systems which demonstrates chaotic behavior, where $N(t)$ is the value of chaotic variable in t^{th} iteration and R shows the bifurcation parameter of the system.

In Fig. 1, a comparison is made between chaotic numbers and random numbers. The above formula was coded in Visual C^{++} to generate 200 chaotic numbers between 0 and 1. Here, R and N are initialized to 4 and 0.1, respectively, while rand() function is used to generate 200 random numbers. The function rand() returns a random value between 0 and RAND_MAX, where RAND_MAX is maximum value that can be stored by an integer variable. The numbers were divided by RAND_MAX to get 200 random numbers between 0 and 1. From Fig. 1, it can be observed that chaotic numbers have higher degree of disorder which facilitates high diversity in the particles and thus helps the algorithm to converge rapidly towards the solution.

3.2 Mutation operator adopted from genetic algorithm

In particle swarm optimization, although the rate of convergence is good due to fast information flow among the solution vectors, its diversity decreases very quickly in the successive iterations resulting in a sub-optimal solution. Researchers have suggested methods to overcome this drawback with varying degrees of success [29, 30]. It was found, when the particles become immobile and stuck at a point, the point was very tightly clustered and the velocities of the particles had almost reduced to zero. Due to negligible velocities, the equations were unable to generate any new solution, which often cause the swarm to stagnate at some suboptimal point or at some local optima.

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation is an important part of the genetic search as it helps to prevent the population from stagnating at any local optima. There are two types of mutations: (1) mutation

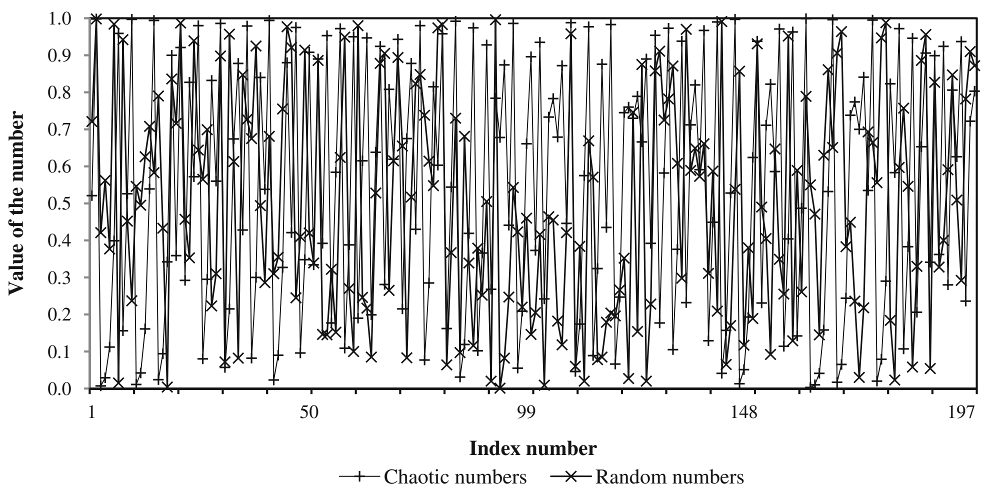


Fig. 1 Comparison between random numbers and chaotic numbers

of global best and (2) mutation of local best. Since all the particles move towards the global best, it is possible to lead the swarm away from a current location by mutating a single individual if the mutated individual is better than the current global best. Looking at the individual components of solution vectors corresponding to the global best function values revealed that it was often only a few components which had not converged to their global optimum values. This suggested the possibility of mutating a single component only of a solution vector. The latter approach introduces diversity by mutating few individuals in the swarm.

In this paper, a mutation operator is introduced which mutates some particles selected randomly from the swarm. Mutation is not carried out each time. A function, $N(0, \mathrm{MAXT})$ returns an integer greater than or equal to 0 and less than MAXT. If the number of iterations with no diversity in the solution exceeds this value, mutation is carried out. There are two reasons for adopting this method: (1) if there is no premature convergence, the execution of PSO will not be affected; (2) the algorithm will not increase computational overhead much. The mutation operator swaps between two positions of a particle randomly. Then, permutations are generated for new positions.

3.3 Mutation algorithm

```
//DELTA: number of iterations with no diversity.
//MAXT: maximum number of iterations with no diversity.
Initialize $t=0$
initialize the swarm
evaluate each particle using encoding scheme
while (not-termination-condition)
$t = t + 1$
update swarm according to formulae 1 and 2
if (DELTA>$N(0, \mathrm{MAXT})$)
do mutation
end if
evaluate the swarm
End
```

4 Problem description

In a flexible flow shop problem, a set of n independent jobs are available simultaneously to be processed sequentially on different stages of a production facility as illustrated in Fig. 2. Each job j has its fixed standard processing time for every stage s, s ∈{1,2,3…..K}, and at each stage, there is a set of unrelated parallel machines M ∈{1,2,3…..m} where some production stages may have only one machine but at least one production stage must have multiple machines. The multiprocessor task scheduling is defined as each task within a job can be processed by several machines (processors) simultaneously at each stage. The tasks are related to each other such that some tasks cannot start to process while its precedence is not accomplished. After a task is processed, its successor task may be processed but only after a predefined time. Each processor can process just one task simultaneously.

The jobs arrive at first stage where the corresponding operations are to be performed and the jobs are delivered to the next stage for the completion of succeeding operations. The jobs have to pass through all the stages sequentially. Such a flexible flow shop problems assumes a static and deterministic scheduling environment. There are no precedence relationships between jobs. Preemption is not allowed, i.e., once an operation is started; it must be completed without interruption. The operations of a job have to be realized sequentially without overlapping between stages. Job splitting is also not permitted. Each job can be processed by only one machine at each machine stage. The processing times of a job are equal on all machines at the same machine stage. The objective of scheduling is to assign jobs to the machines at the corresponding stages and determine the processing sequences on the machines so that the makespan (C_{max}), i.e., the maximum completion time is minimized.

5 Problem representation

In flexible flow shop scheduling problem, we have used a real number encoding whose fractional part is used to sort

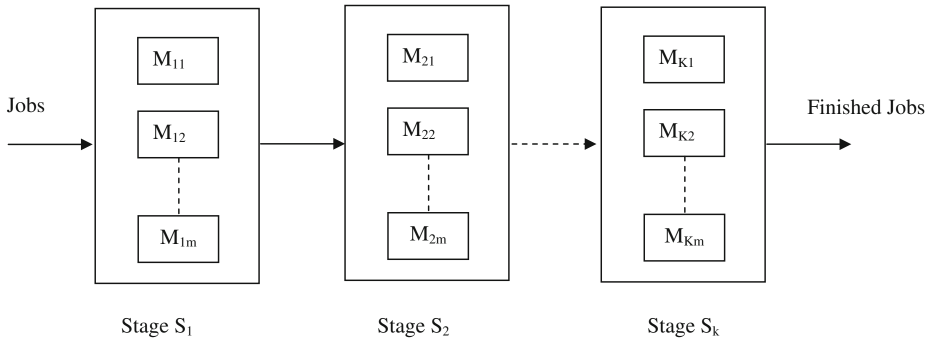


Fig. 2 A flexible flow shop environment

the jobs assigned to each machine and whose integer part is the machine number to which the job is assigned. The position of the particle represented by a real number is used for encoding. Suppose an FFSP has n jobs and three processing stages. Stages 1, 2, and 3 are having two, three, and four machines, respectively. We generate n real numbers randomly by uniform distribution between [1, 1+m_K](K=1,2,3) for each stage where m_K is the number of machines in K stage. For example, we have four jobs to be processed. The processing times $\Pr = \begin{bmatrix} 7 & 4 & 2 \\ 2 & 7 & 3 \\ 5 & 3 & 1 \\ 4 & 5 & 2 \end{bmatrix}$ are given. The initial positions of the particle are generated as four real numbers within the range of 1–3 for the first stage. The real part of the position values is used for ensuring machine number on which the job is to be processed and fractional part is used to sequence the jobs on the same machine. For stage 1, the position values of four jobs is given as P=[1.05, 2.33, 1.45, 2.67]. At stage 1, jobs 1 and 3 are assigned to machine 1 and jobs 2 and 4 are assigned to machine 2. The process order of jobs to be scheduled on the same machine depends on the value of fractional parts. The job is sequenced according to the ascending order of the fractional part which is processed by the same machine. The order of jobs to be scheduled on machine1 is job 3 followed by job 1 because the fractional part of job 3 is greater than the fractional part of job 1. From stage 2 to the last stage, the process order of jobs to be scheduled on the same machine depends on the value of fractional parts and the completion time of the former stage. If the completion time of the former stage is the same, we can compare the values of particle position. The job whose value of particle position is smaller will be first processed. If the values are also equal, we can ensure the job processing sequence randomly. The encoding scheme along with the Gantt chart for a possible solution is shown in Fig. 3.

6 Proposed algorithm

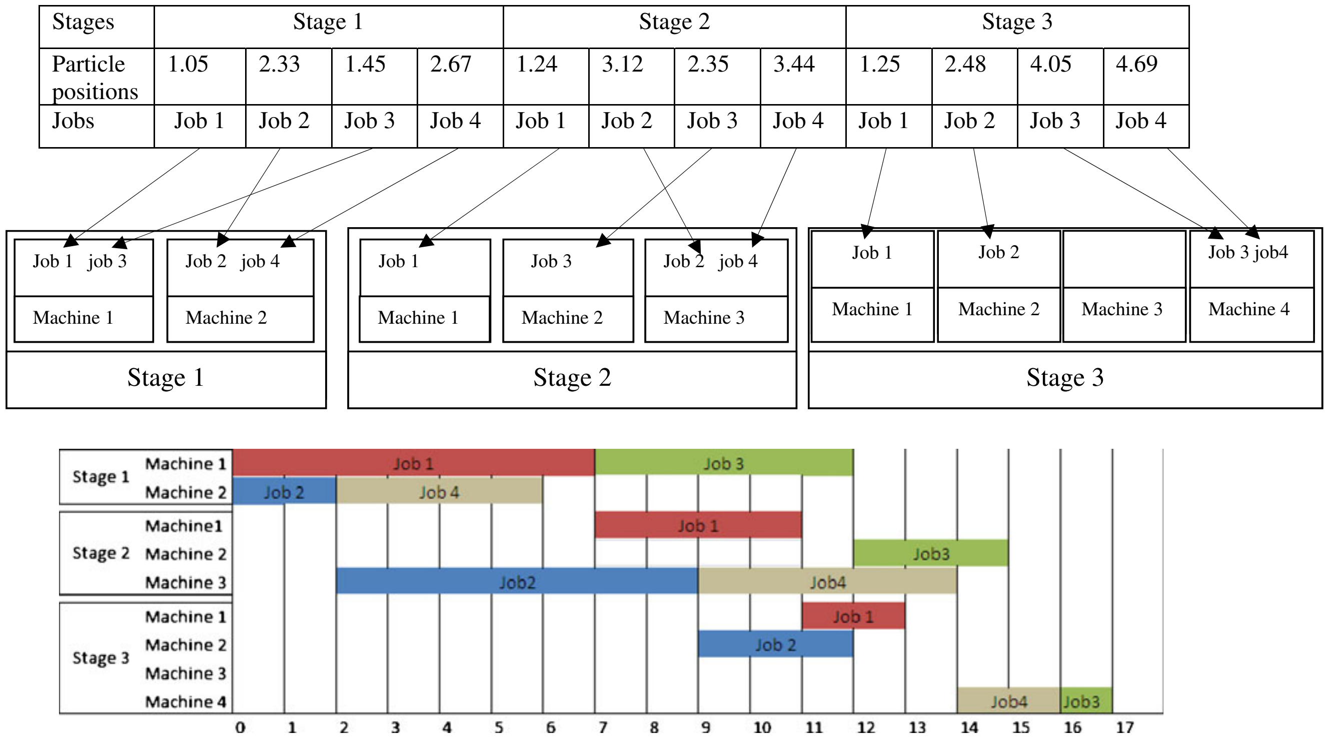
Step 1. Initialize the parameters such as population size, maximum iteration, decrement factor, inertia weight, social and cognitive parameters.

Step 2. Input number of jobs, number of stages, number of machines at each stage, and processing times.

Step 3. Generate the initial position values of the particle uniformly: $x_{ij}^{t} = x_{\min} + (x_{\max} - x_{\min}) \times N(0,1)$, where $x_{\min}$=0.0, $x_{\max}$=4.0, and N(0,1) is a chaotic number between 0 and 1. Generate initial velocities of the particle $v_{ij}^{t} = v_{\min} + (v_{\max} - v_{\min}) \times N(0,1)$, where $v_{\min}$=−4.0, $v_{\max}$=4.0, and N(0,1) is a chaotic number between 0 and 1.

Step 4. Get the schedule using encoding scheme as mentioned in Section 5.

Step 5. Evaluate each particle's fitness (makespan).



Stages	Stage 1				Stage 2				Stage 3			
Particle positions	1.05	2.33	1.45	2.67	1.24	3.12	2.35	3.44	1.25	2.48	4.05	4.69
Jobs	Job 1	Job 2	Job 3	Job 4	Job 1	Job 2	Job 3	Job 4	Job 1	Job 2	Job 3	Job 4

Fig. 3 Problem representation

Step 6. Find out the personal best (pbest) and global best (gbest).

Step 7. If no progress is observed in pbest value for an elapsed number of iterations DELTA, carry out mutation of a particle using the mutation strategy as outlined in Section 3.3 provided DELTA is greater than a chaotic number between 0 and maximum time of no progress (MAXT).

Step 8. Update velocity, position and inertia weight by using Eqs. 1, 2, and 3.

Step 9. Terminate if maximum number of iterations is reached and store the gbest value. Otherwise, go to step 3.

Step 10. End

7 Results and discussion

The computational study aims to analyze the performance of PSO to minimize the makespan for the flexible flow shop scheduling with multiprocessor task problems. The algorithm was coded in visual C^{++} and ran on a PC Pentium 4 processor with 3 GHz and 1 GB memory. The example problem, shown in Fig. 3, results a makespan of 17. The same problem when solved using PSO gives a makespan of 15 at 5,000 iterations with population size of 15. The inertial weight (w) is set at 0.7, social and cognitive parameters (c_1 and c_2) is 0.15, and decrement factor (α) is 0.6. The Gantt chart for the solution is shown in Fig. 4.

The benchmark problems are taken from Oguz [31]. The size of the problem ranges from 5 jobs×2 stages to 100 jobs×8 stages. Processing times were generated randomly in the range (1, 100). There are two types of problem—P and Q types. In type P problems, the number of processors available at various stages is randomly selected from the set (m_i=2…5), whereas in type Q problems, it is fixed with m_i=5 for all stages. The P50S8T05 problem is defined as 50 jobs, 8 stages, 05 problem index, and P-type problem. In P1HS8T05 problems, 1HS represents 100 jobs. In order to validate the performance of the proposed method, the results are compared with the earlier studies of Oguz et al. [7], Engin et al. [3], and Kahraman et al. [32]. In this paper, only P-type problems are considered. The results are represented in terms of percentage deviation (PD) of the solution from the lower bound (LB). Percentage deviation is defined as in the following:

$$\mathrm{PD} = \frac{\text{Best } C_{\max} - \text{lowerbound}}{\text{lowerbound}} \times 100 \tag{5}$$

The LB of the makespan was proposed by Oguz et al. [33] and is defined by the following relation:

$$\mathrm{LB} = \max_{i\in M}\left\{\min_{j\in J}\left\{\sum_{k=1}^{i-1} \mathrm{p_{kj}}\right\} + \frac{1}{m_i}\sum_{j\in J}\mathrm{p_{ij}size_{ij}} + \min_{j\in J}\left\{\sum_{k=i+1}^{m} \mathrm{p_{kj}}\right\}\right\} \tag{6}$$

where m_i is the number of processors available for tasks corresponding to stage i and $\mathrm{size_{ij}}$ be the number of processors required in order to process job j at stage i. $\mathrm{P_{ij}}$ be the processing time of job j at stage i.

In Table 1, column named as problem represents the name of the problem family, GA represents the PD results of Oguz et al.'s genetic algorithm [7], PGA represents the PD results of Kahraman et al.'s parallel greedy algorithm [32], and efficient GA represents the PD results of Engin et al.'s efficient GA [3] and the proposed PSO algorithms with PD.

For 120 problems solved here, it is observed from Table 1 that the proposed PSO algorithm gives smaller PD in 68 problems over GA, 58 problems over PGA, and 51 problems over efficient GA. Average percentage deviation (APD) of the proposed PSO is also compared with GA, PGA, and efficient GA for P-type flexible flow shop problem. The APD is defined as follow.

$$\mathrm{APD} = \frac{\sum_{L=1}^{I}\mathrm{PD(L)}}{I}$$

where I is the total number of problems and L stands for index of the problem. The comparison results are presented in Table 2. The APD for GA, PGA, and efficient GA are 3.559, 3.470, and 3.147, respectively. APD for PSO is obtained as 2.961 which happens to be superior as compared

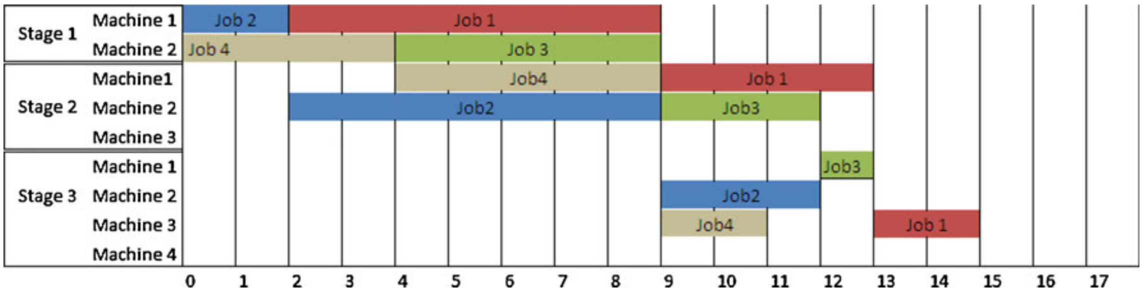


Fig. 4 The final scheduling result in the example problem

Table 1 The computational results

	PD					PD			
Problems	GA [7]	PGA [32]	Efficient GA [3]	PSO	Problems	GA [7]	PGA [32]	Efficient GA [3]	PSO
P10S2T01	0	0	0	0	P10S5T01	4.53	4.878	4.53	1.744
P10S2T02	0	0	0	0	P10S5T02	6.022	6.022	6.022	1.748
P10S2T03	5.96	5.96	5.96	8.7	P10S5T03	7.54	7.54	7.54	2.673
P10S2T04	0	0	0	0	P10S5T04	8.609	11.755	11.755	17.79
P10S2T05	0	0	0	0	P10S5T05	3.684	3.684	3.684	2.636
P10S2T06	0	0	0	0	P10S5T06	0	0	0	0
P10S2T07	0	0	0	0	P10S5T07	8.21	8.21	8.21	32.7
P10S2T08	0	0	0	10	P10S5T08	2.742	2.258	2.258	28.065
P10S2T09	0	0	0	0	P10S5T09	9.076	9.076	9.076	3.81
P10S2T10	0	0	0	0	P10S5T10	10.417	12.179	12.66	11.2
P20S2T01	1.997	1.69	2.611	8.089	P20S5T01	1.567	1.567	1.567	0
P20S2T02	2.368	1.275	1.639	6.98	P20S5T02	7.165	6.858	7.165	0
P20S2T03	0	0	0	0	P20S5T03	2.561	6.17	2.561	10.5
P20S2T04	0	0	0	6.8	P20S5T04	2.268	0.113	0	0
P20S2T05	0	0	0	0	P20S5T05	2.313	3.049	1.682	8.014
P20S2T06	0	0	0	8.577	P20S5T06	2.65	2.479	1.88	0
P20S2T07	0.341	0.341	0.341	0	P20S5T07	0	0	0	1.434
P20S2T08	0.278	0.278	0.278	4.02	P20S5T08	3.519	3.519	3.128	14.33
P20S2T09	0	0	0	0	P20S5T09	0	0	0	0
P20S2T10	0	0	0	1.76	P20S5T10	7.165	7.165	6.858	0
P50S2T01	1.753	1.088	0.605	1.046	P50S5T01	0.911	0.835	0.911	0
P50S2T02	0	0	0	0	P50S5T02	4.539	0.532	1.099	0
P50S2T03	0.59	0.215	0	17.2	P50S5T03	0.998	0.998	0.998	0
P50S2T04	1.238	1.423	0.681	12.027	P50S5T04	0.618	4.325	2.523	17.507
P50S2T05	0.353	0.588	0.235	3.166	P50S5T05	2.577	0	0	0
P50S2T06	0	0	0	0	P50S5T06	0	0	0	0
P50S2T07	2.716	2.13	0.692	19.87	P50S5T07	1.1	0.477	0.513	0
P50S2T08	0	0	0	0	P50S5T08	2.447	0.745	1.099	0
P50S2T09	0	0	0	0	P50S5T09	5.096	0.668	0	0
P50S2T10	0.262	0.052	0.052	3.36	P50S5T10	0.271	1.264	0.587	3.324
P1HS2T01	0.534	0.507	0.721	0	P1HS5T01	3.493	0.056	0	0
P1HS2T02	0	0	0	0	P1HS5T02	1.543	0	0	11.76
P1HS2T03	0.814	0.603	0.693	2.84	P1HS5T03	3.819	2.272	0.674	1.8
P1HS2T04	0	0	0	0	P1HS5T04	1.425	1.548	1.13	2.994
P1HS2T05	0	0	0	0	P1HS5T05	2.347	0	0	0
P1HS2T06	0	0	0	0	P1HS5T06	3.591	4.488	2.924	5.56
P1HS2T07	0.926	0.538	0.657	0.81	P1HS5T07	0.3	0	0	0
P1HS2T08	0.02	0	0	0	P1HS5T08	0.673	0	0	0
P1HS2T09	0.105	0.394	0.683	0.116	P1HS5T09	1.379	0	0	0.399
P1HS2T10	0.097	0.242	0.097	0	P1HS5T10	3.248	3.389	2.994	3.5019
P10S8T01	21.268	23.662	21.268	9.59	P50S8T01	2.709	6.578	5.288	0
P10S8T02	26.179	28.455	31.87	0	P50S8T02	2.75	2.75	0	1.72
P10S8T03	20.027	21.786	20.027	12.71	P50S8T03	0.936	1.161	0.787	0
P10S8T04	13.091	14.038	17.35	7.57	P50S8T04	4.76	3.917	2.696	0
P10S8T05	1.902	4.212	4.212	0	P50S8T05	3.639	2.691	0.986	0
P10S8T06	0.409	0.409	0.409	0	P50S8T06	1.875	1.641	0.313	0.351
P10S8T07	24.757	26.214	30.097	0.91	P50S8T07	5.436	3.701	2.544	0.1126

Table 1 (continued)

Problems	GA [7]	PD PGA [32]	Efficient GA [3]	PSO	Problems	GA [7]	PD PGA [32]	Efficient GA [3]	PSO
P10S8T08	19.479	21.933	19.479	8.282	P50S8T08	2.434	1.917	1.881	1.659
P10S8T09	0.83	2.075	2.075	0	P50S8T09	4.76	4.465	3.412	0
P10S8T10	19.589	20	19.589	12.876	P50S8T10	5.371	4.726	3.187	6.4804
P20S8T01	8.163	9.448	9.599	0	P1HS8T01	2.877	1.509	0	0
P20S8T02	0	0	0	0	P1HS8T02	3.568	0.801	0.157	1.461
P20S8T03	2.573	4.117	2.573	0	P1HS8T03	1.987	1.472	0.644	0
P20S8T04	1.593	3.805	7.08	0	P1HS8T04	2.149	0.498	0.103	0
P20S8T05	3.152	2.641	2.3	2.214	P1HS8T05	1.944	0.731	0.019	0
P20S8T06	8.163	9.675	4.913	0	P1HS8T06	3.422	2.907	1.104	4.839
P20S8T07	23.795	27.487	28.821	0	P1HS8T07	2.426	1.203	0.019	1.6806
P20S8T08	3.791	2.729	3.791	0	P1HS8T08	7.294	8.451	5.584	0
P20S8T09	0	0	0	0	P1HS8T09	1.951	1.334	0.02	0.11
P20S8T10	4.117	4.803	3.945	0	P1HS8T10	3.838	0.985	0.185	0

to other algorithms. The performance of GA is worst among all the algorithms.

The improvement rate for total average APD using PSO is defined as follows:

$$\text{Improvement rate}(\%) = \frac{\left(\text{APD}_{\text{GA,PGA,EGA}} - \text{APD}_{\text{PSO}}\right)}{\text{APD}_{\text{GA,PGA,EGA}}}$$

On the total average APD of PSO, an improvement of 16.88% with respect to GA, 14.66% with respect to PGA, and 5.923% with respect to efficient GA has been achieved for P-type problems considered in the study.

Experiments are carried out to establish effect of various parameters used in the proposed PSO algorithm on solution quality. Three characteristics are used to represent a problem such as number of jobs, the number of stages, and the number of machines at each stage. The problem sizes vary from 25 jobs×3 stages to 30 jobs×5 stages. The processing times of jobs are uniformly generated over the interval [5, 50]. An experiment has been conducted for a five-stage, 30 jobs, and five machines at each stage problem in which population size (p) varies from 10 to 20 keeping other parameters fixed at w=0.7, c_1 and c_2=0.15, and α=0.6. From Fig. 5, it is evident that the algorithm finds the optimal solution quickly when the number of population size increased. Here, we got the near-optimal solution only at the 1,000th iteration when the population size is 20. But after 15,000 iterations, it is close to the near-optimal solution when the population size is 10 and 15.

Table 2 APD of the proposed PSO compared with GA, PGA, and efficient-GA

Stage	Job	APD GA [7]	PGA [32]	Efficient GA [3]	PSO
2	10	0.596	0.596	0.596	1.870
2	20	0.489	0.359	0.487	4.023
2	50	0.691	0.550	0.227	3.662
2	100	0.250	0.228	0.285	0.377
5	10	6.820	6.560	6.574	10.237
5	20	2.920	3.092	2.484	3.428
5	50	1.855	0.984	0.773	2.083
5	100	2.188	1.175	0.772	2.601
8	10	14.753	16.278	16.37	5.193
8	20	5.535	6.471	6.302	0.221
8	50	3.467	3.355	2.109	1.032
8	100	3.146	1.989	0.784	0.809
Total average (%)		3.559	3.470	3.147	2.961

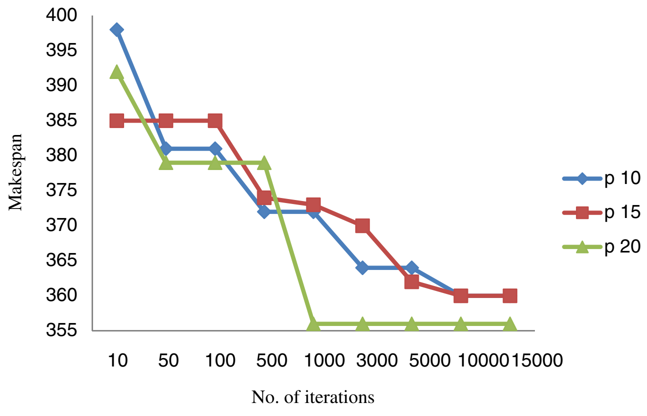


Fig. 5 The performance of the PSO algorithms with variation of population size

Figure 6 shows the makespan values for a 30-job problem varying the stages from 3 to 5 in which number of machines at each stage is fixed at 5. The population size is fixed at 15 and other parameters are set at w=0.7, c_1 and c_2= 0.15, and α=0.6. The graph indicates that makespan increases with number of stages.

Figure 7 shows the makespan values with varying number of machines in each stage. Here, the experiments were conduct with five-stage and 30-job problem by varying number of machines at each stage from 5 to 15. The population size is fixed at 15 and other parameters are set at w= 0.7, c_1 and c_2=0.15, and α=0.6. The graph shows that makespan decreases as the number of machine in each stage increases due to reduction of idle time. However, duplicating machines at each stage involves investment implications.

The convergence curve is shown for a 30-job, five-stage having 10 machines at each stage problem. The population size is fixed at 15 and other parameters are set at w=0.7, c_1 and c_2=0.15, and α=0.6. It can be observed from Fig. 8 that the solution converges towards best value at a faster rate. The best value of makespan is obtained at 3,000 iterations.

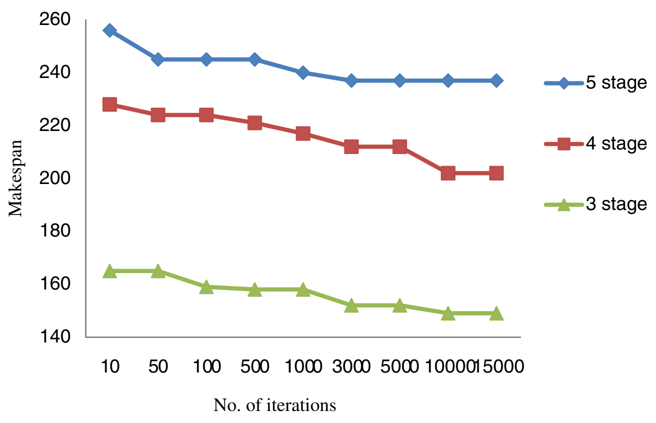


Fig. 6 The performance of the PSO algorithms with variation of stage size

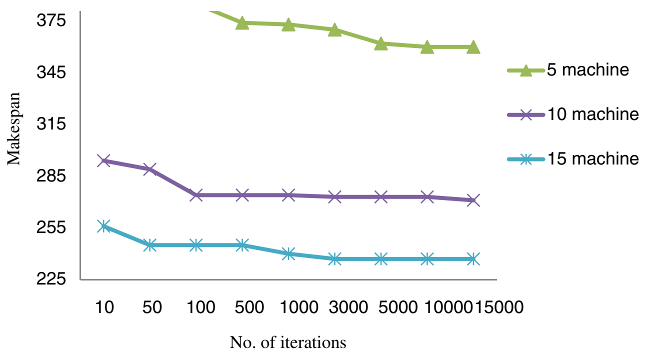


Fig. 7 The performance of the PSO algorithms with variation of stage size

8 Conclusion

In this paper, flexible flow shop scheduling problem which is NP-hard in the strong sense was considered, and an efficient particle swarm optimization on the value of an optimal schedule was developed. A mutation operator used in genetic algorithm is introduced in PSO to avoid premature convergence and improve solution diversity. Solution diversity is also improved through the use of chaotic numbers (logistic map) instead of random numbers. Thereby, computational efforts can be reduced by a large extent. The proposed approach is tested on a set of 120 instances of P-type problems taken from the literature [31]. The comparison was made with the flexible flow shop scheduling with multiprocessor task problems from Oguz et al.'s [7] genetic algorithms, Kahraman et al.'s [32] parallel greedy algorithm, and Engin et al.'s [3] efficient GA. The obtained results are encouraging in that the proposed algorithm gives smaller percentage deviation from lower bound for 68 problems over GA, 58 problems over PGA, and 51 problems over efficient GA. Based on the total average percentage deviation (APD), PSO results in an improvement of 16.88% with respect to GA, 14.66% with respect to PGA, and 5.923% with respect to efficient GA for P-type problems considered in the study. The advantage of PSO algorithm

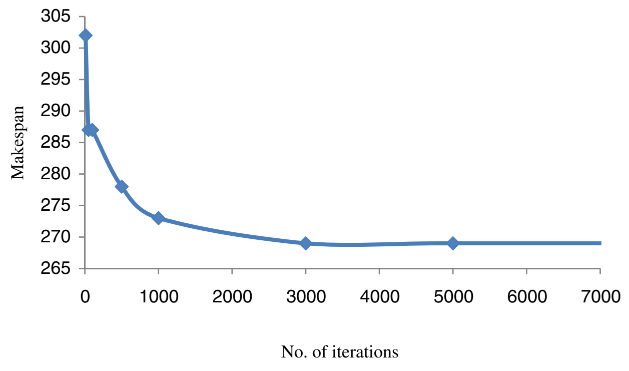


Fig. 8 The convergence curve of the PSO algorithm

lies in the fact that it requires less number of parameters to be controlled at arrive at good solutions as compared to other population-based methods. The inherent drawbacks of PSO have been effectively tackled through use of mutation inspired from genetic algorithm. The chaotic number used in the work provides solution diversity and reduces computational burden. The proposed PSO approach is a good problem solving technique for a scheduling problem. For future research, hybrid PSO algorithm may be used for some other industrial problems.

References

1. Gupta JND (1988) Two stage hybrid flow shop scheduling problem. J Oper Res Soc Jpn 39(4):359–364
2. Hoogeveen JA, Lenstra JK, Veltman B (1996) Minimizing the makespan in a multiprocessor flow shop is strongly NP-hard. Eur J Oper Res 89:172–175
3. Engin O, Ceran G, Yilmaz MK (2011) An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems. Appl Soft Comput 11:3056–3065
4. Arthanari TS, Ramamurthy KG (1971) An extension of two machines sequencing problem. Eur J Oper Res 8:10–22
5. Brah SA, Hunsucker JL (1991) Branch and bound algorithm for flow shop with multiple processors. Eur J Oper Res 51:88–89
6. Moursli O, Pochet Y (2000) A branch and bound algorithm for the hybrid flow shop. Int J Prod Econ 64:113–125
7. Oguz C, Ercan F (2005) A genetic algorithm for hybrid flow shop scheduling with multiprocessor tasks. Journal of Scheduling 8:323–351
8. Kennedy J, Eberhart R (1995) Particle swarm optimization. Proceedings of IEEE Int. Conference on Neural Network 1942–1948
9. Xia WJ, Wu ZM (2006) A hybrid particle swarm optimization approach for the job shop scheduling problem. Int J Adv Manuf Tech 29:360–366
10. Xia WJ, Wu ZM (2005) An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. Comput Ind Eng 48(2):409–425
11. Jayamohan MS, Rajendran C (2000) A comparative analysis of two different approaches to scheduling in flexible flow shops. Prod Plann Contr 2(6):572–580
12. Kurz ME, Askin RG (2003) Comparing scheduling rules for flexible flow lines. Int J Prod Econ 85(3):371–388
13. Bertel S, Billaut JC (2004) A genetic algorithm for an industrial multi-processor flow shop scheduling problem with recirculation. Eur J Oper Res 159(3):651–662
14. Xiao W, Hao P, Zhang S, Xu X (2000) Hybrid flow shop scheduling using genetic algorithms. In Proceedings of the 3rd World Congress on Intelligent Control and Automation 537–541
15. Kurz ME, Askin RG (2004) Scheduling flexible flow lines with sequence-dependent setup times. Eur J Oper Res 159(1):66–82
16. Bean JC (1994) Genetic algorithms and random keys for sequencing and optimization. INFORMS J Comput 6(2):154–160
17. Ruiz R, Maroto C (2006) A genetic algorithm for hybrid flow shops with sequence dependent setup times and machine eligibility. Eur J Oper Res 169(3):781–800
18. Yoshida H, Kawata K, Fukuyama Y, Nakanishi Y (2001) A particle swarm optimization for reactive power and voltage control considering voltage security assessment. IEEE Trans Power Syst 15(4):1232–1239
19. Van den Bergh F, Engelbecht AP (2000) Cooperative learning in neural networks using particle swarm optimizers. S Afr Comput J 26(6):84–90
20. Brandstatter B, Baumgartner U (2002) Particle swarm optimization mass-spring system analogon. IEEE Trans Magn 38 (2):997–1000
21. Salman A, Ahmad I, Al-Madani S (2002) Particle swarm optimization for task assignment problem. Microprocessors Microsystems 26(8):363–371
22. Onwubolu GC, Clerc M (2004) Optimal operating path for automated drilling operations by a new heuristic approach using particle swarm optimization. Int J Prod Res 42(3):473–491
23. Abido MA (2002) Optimal power flow using particle swarm optimization. Electr Power Energy Syst 24(7):563–571
24. Hu X, Eberhart R (2002) Multiobjective optimization using dynamic neighborhood particle swarm. In Proceedings of the IEEE Congress on Evolutionary Computation, CEC'2002, IEEE Press, Honolulu, Hawaii, New Jersey:1677–1681
25. Mostaghim S, Teich J (2003) Strategies for finding good local guides in multi-objective particle swarm optimization. In Proceedings of the IEEE Swarm Intelligence Symposium, SIS'2003, 2003 April 24–26, IEEE Press, Indianapolis, Indiana, New Jersey: pp. 26–33
26. Wang LF, Singh C (2007) Environmental/economic power dispatch using a fuzzified multi-objective particle swarm optimization algorithm. Electr Power Syst Res 77(12):1654–1664
27. Caponetto R, Fortuna L, Fazzino S, Xibilia GM (2003) Chaotic sequences to improve the performance of evolutionary algorithm. IEEE Trans Evol Comput 7(3):289–304
28. Yao X, Liu Y (1999) Evolutionary programming made faster. IEEE Trans Evol Comput 3(2):82–102
29. Riget J, Vesterstroem JS. A diversity-guided particle swarm optimiser—the ARPSO. Technical Report 2002–02, Department of Computer Science, University of Aarhus
30. Krink T, Lovbjerg M (2002) The life cycle model: combining particle swarm optimization, genetic algorithms and hill climbers. In Proceedings of the 7th International Conference on Parallel Problem Solving from Nature 7:621–630
31. Oguz C (2006) Data for hybrid flow shop scheduling with multiprocessor tasks, KOC University Available at http://home.ku.edu.tr/~coguz/
32. Kahraman C, Engin O, Kaya I, Ozturk RE (2010) Multiprocessor task scheduling in multistage hybrid flow-shops: a parallel greedy algorithm approach. Appl Soft Comput 10:1293–1300
33. Oguz C, Zinder Y, Do VH, Janiak A, Linchtenstein M (2004) Hybrid flow shop scheduling problems with multiprocessor task systems. Eur J Oper Res 152:115–131