Int J Adv Manuf Technol (2008) 39:931–942
DOI 10.1007/s00170-007-1284-5

ORIGINAL ARTICLE

Modified particle swarm optimization for solving machine-loading problems in flexible manufacturing systems

Sandhyarani Biswas · S. S. Mahapatra

Received: 24 July 2007 / Accepted: 10 October 2007 / Published online: 7 November 2007
© Springer-Verlag London Limited 2007

Abstract In flexible manufacturing systems (FMSs), the loading problem is considered as a vital pre-release decision because its operational effectiveness largely depends on a good quality solution to the loading problem. Difficulties arise in obtaining optimal solutions to such problems because of its combinatorial and NP-hard nature. In the past, numerous techniques have been suggested and found to be efficient, but they take long computational times when the problem size increases. In order to address the above issues, a meta-heuristic approach based on particle swarm optimization (PSO) has been proposed in this paper to improve the solution quality and reduce the computational effort. However, PSO has the tendency to suffer from premature convergence. Therefore, the PSO algorithm has been modified through the introduction of a mutation operator to improve efficiency of the algorithm. The proposed algorithm attempts to minimize the system unbalance while satisfying the technological constraints, such as the availability of machining time and tool slots. The proposed algorithm produces promising results in comparison to existing methods for ten benchmark instances available in the FMS literature.

Keywords Flexible manufacturing systems · Machine-loading problem · Particle swarm optimization · Mutation · System unbalance

S. Biswas (✉) · S. S. Mahapatra
Department of Mechanical Engineering,
National Institute of Technology,
Rourkela 769008, India
e-mail: sandhya_biswas@yahoo.co.in

S. S. Mahapatra
e-mail: mahapatrass2003@yahoo.com

1 Introduction

The advent of global economy and trade has resulted in competition that has led enterprises to face a dynamic environment. Such an environment is characterized by a large volume of uncertainty, such as rapid market changes, increased product variety, competitive prices, and short product life cycles. Therefore, it is of prime importance to introduce flexible manufacturing systems (FMSs) so that these uncertainties can be handled in an effective manner. According to Stecke [1], an FMS is characterized as an integrated, computer-controlled complex arrangement of automated material-handling devices and numerically controlled (NC) machine tools that can simultaneously process medium-sized volumes of a variety of part types. The highly integrated FMS offers the opportunity to combine the efficiency of a transfer line and the flexibility of a job shop to best suit the batch production of mid-volume and mid-variety of products. However, flexibility has a cost, and the capital investment sustained by firms to acquire such systems is generally very high. Therefore, particular attention must be paid to the proper planning of an FMS during its development phase in order to evaluate the performance of the system and justify the investment incurred. Prior to production, careful operational planning is essential to establish how well the system interacts with the operations over time. Hence, the successful operation of an FMS requires more intense planning compared to any conventional production system.

The decisions related to FMS operations can be broadly divided into pre-release and post-release decisions. Pre-release decisions include the FMS operational planning problem that deals with the pre-arrangement of jobs and tools before the processing begins, whereas post-release decisions deal with the scheduling problems [1]. Pre-release

decisions, e.g., machine grouping, part type selection, production ratio determination, resource allocation, and loading problems, must be solved while setting up an FMS. Amongst pre-release decisions, machine loading is considered as one of the most vital production planning problems because the performance of the FMS largely depends on it. Loading problems, in particular, deal with the allocation of jobs to various machines under technological constraints, with the objective of meeting certain performance measures, hence, it is considered as a combinatorial optimization problem and happens to be NP-hard in nature. Numerous methods based on mathematical, heuristics, meta-heuristics, and simulation have been suggested by researchers in the pursuit of obtaining quality solutions to loading problems and reduce computational burden. But these approaches are barely capable of producing optimal/near-optimal solutions or require excessive computational efforts to arrive at good quality solutions. In order to alleviate these difficulties, an attempt has been made in this paper to propose an algorithm based on particle swarm optimization (PSO) to solve the machine-loading problem of a random FMS with the objective of the minimization of system unbalance while satisfying the constraints related to the available machining time and tool slots. However, PSO has the inherent drawback of trapping at a local optimum due to the large reduction in velocity values as the iterations proceed and, hence, reduces the solution variety. This drawback has been addressed effectively by incorporating mutation, a commonly used operator in genetic algorithms, to improve the solution quality. In addition, the study analyzes the effect of the underloading/overloading of machines on the solution quality in a loading problem.

The remainder of this paper is organized as follows. Section 2 gives a brief review of the past literature on the loading problem. Section 3 formally defines the problem studied in this paper, along with the objectives and assumptions made to solve the problem. The proposed algorithm based on PSO is presented in Sect. 4. In Sect. 5, the results of benchmark problems from the open literature are compared with the proposed method to illustrate its advantage over the other methods. Finally, conclusions drawn from this study are summarized and directions for future research are outlined in Sect. 6.

2 Literature review

The pioneering work in FMS planning problems formulated as a non-linear 0–1 mixed-integer programming (MIP) problem was proposed by Stecke and Solberg [2] and was subsequently improved by Stecke [1]. Berrada and Stecke [3] proposed a branch-and-bound algorithm to balance the workloads on machines. Sarin and Chen [4] used an MIP approach to determine part routings through the machines and to allocate appropriate cutting tools to each machine for achieving minimum machining cost, which are dependent on the machine–tool combinations. Chen and Chung [5] discussed the effects of loading and routing decisions on the performance of an FMS. Atmani and Lashkari [6] and Gamila and Motavalli [7] presented a linear 0–1 integer programming model for machine tool assignment and operation allocation to determine an optimal plan by minimizing the total costs and the time of operations, respectively, considering material handling and setups. A few researchers, such as Shanker and Tzen [8], O'Grady and Menon, [9], and Liang and Dutta [10–12], have attempted to solve the loading problem with a bi-criterion objective for the machine-loading problem. Although analytical- and mathematical-programming-based methods are robust in applications, they tend to become impractical when the problem size increases. This has motivated researchers to develop fast and effective heuristics for solving loading problems in large-sized FMSs. Stecke and Tallbot [13] have reported that mathematical approaches may not provide practical solutions to real-time applications because they require a large amount of computational effort. Hence, they have proposed heuristic algorithms for the minimization of part movements that balances the load on machines of equal size. Mukhopadhyay et al. [14] suggested a heuristic-based approach to solve the loading problem in FMSs by developing the concept of essentiality ratio for the maximization of throughput and the minimization of system unbalance simultaneously. Tiwari et al. [15] proposed a heuristic approach that uses fixed predetermined job ordering rules as the input while solving loading problems. However, it has been established by Stecke and Solberg [2], Shanker and Tzen [8], and Moreno and Ding [16] that the shortest processing time (SPT) rule works better in comparison to other rules, e.g., longest processing time (LPT), first-in-first out (FIFO), and last-in-first-out (LIFO). Srinivas et al. [17] addressed the FMS machine-loading problem through winner determination using the combinatorial auction process so that both the minimization of system unbalance and the maximization of throughput can be achieved simultaneously. Nagarjuna et al. [18] proposed a heuristic based on a multi-stage programming approach for minimizing the system unbalance while satisfying constraints such as the availability of tool slots and machining time.

The major limitation of a heuristic is its inability to estimate the results in a new or completely changed environment, as they are generally rule-based and mostly rely on empirical data. Therefore, numerous researchers have used meta-heuristic approaches for solving the machine-loading problem. Mukhopadhyay et al. [19] proposed a perturbation scheme known as a "modified

insertion scheme" for generating new job sequences in their simulated annealing (SA) approach. Genetic algorithm (GA)-based approaches for loading problems is found to ensure an optimal solution and to be less computationally intensive. Kumar and Shanker [20], Tiwari and Vidyarthi [21], and Swarnkar and Tiwari [22] have addressed machine-loading problems having the bi-criterion objectives of minimizing system unbalance and maximizing the throughput using a hybrid algorithm based on tabu search (TS) and SA. The main advantage of this approach is that a short-term memory provided by the tabu list can be used to avoid revisiting the solution while preserving the stochastic nature of the SA method. Vidyarthi and Tiwari [23] proposed a fuzzy-based methodology to solve the machine-loading problem in an FMS. The job-ordering determination before loading is carried out by evaluating the membership contribution of each job to its characteristics, such as batch size, essential operation processing time, and optional operation processing time. The operation–machine allocation decisions are made based on the evaluation of the membership contribution of an operation–machine allocation vector. Chan and Swarnkar [24] presented a fuzzy goal programming approach to model the machine tool selection and operation allocation problem. Then, an ant colony optimization (ACO)-based approach is applied to improve the solution further.

When a system cannot be evaluated analytically to obtain a single solution, then simulation may be a good option. In this regard, simulation-based studies have been carried out for testing various dispatching rules by Stecke and Solberg [2] and Gupta et al. [25]. Many simulation studies focused on the determination of optimal operational parameters while designing an FMS [19, 26, 27]. Although simulation is an effective technique for dynamic analysis, it lacks the ability to provide an optimal solution as far as FMS design is concerned. Simulation techniques have another limitation in terms of the number of iterations. A large number of iterations are needed to find the best possible solution and there is always a possibility of returning to some recently found solutions, resulting in a non-optimal solution.

3 Problem description

The loading problem in manufacturing deals with selecting a subset of jobs from a set of all jobs to be manufactured and assigning their operations to the relevant machines in a given planning horizon with technological constraints in order to meet certain performance measures, such as the minimization of system unbalance and the maximization of throughput. System unbalance can be defined as the sum of unutilized or overutilized times on all of the machines available in the system, whereas throughput refers to the summation of the batch size of the jobs that are to be produced during a planning horizon. The minimization of system unbalance is equivalent to the maximization of machine utilization. The processing time and tool slots required for each operation of the job and its batch size are known beforehand. There are two types of operations associated with the part types; essential and optional. Essential operations can be carried out on a particular machine using a certain number of tool slots, while optional operations can be performed on a number of machines with the same or different processing times and tool slots. The FMS under consideration derives its flexibility in the selection of a machine for the optional operation of the job. Generally, the complexity of these problems depends on whether the FMS is of a dedicated type or a random type. A dedicated FMS is designed to produce a rather small family of similar parts with a known and limited variety of processing requirements, while in a random-type system, a large family of parts having a wide range of characteristics with random elements is produced and the product mix is not completely defined at the time of installing the system. This paper addresses the loading problem in a random FMS. Loading in an FMS environment is more complex and difficult to solve. In order to minimize the complexities, the following assumptions are made when analyzing the FMS loading problem:

- Initially, all of the jobs and machines are simultaneously available
- The processing time required to complete an entire job order is known a priori
- The job undertaken for processing is to be completed for all of its operation before considering a new job; this is called non-splitting of the job
- The operation of a job once started on a machine is continued until it is completed
- The transportation time required to move a job between machines is negligible
- The sharing and duplication of tool slots is not allowed.

Different researchers have calculated system unbalance in different ways. Some researchers, such as Mukhopadhyay et al. [14], Swarnkar et al. [22], and Nagarjuna et al. [18], solved the machine-loading problem by considering both the underloading and overloading of machines, while Shanker et al. [28] have not permitted overloading. In order to examine the efficiency of the proposed algorithm, the problem described above is formulated in this paper by considering two cases. In the first case, an optimal solution is obtained without permitting the overloading of machines, whereas the overloading of machines is permitted in the second case. The mathematical formulation of the problem for both cases are

described in the following. The notation used is given as below:

j	Job types, j=1, 2,..., J
m	Machine types, m=1, 2,..., M
o	Operation number of job j, o=1, 2,..., O_j
S_m	Tool slot capacity of machine m
AT_m	Available time on machine m
AS_m	Available tool slots on machine m
B_j	Batch size of job j
O_e	Number of essential operations for job j, O_e=1, 2,..., EO_j
O_o	Number of optional operations for job j, O_o=1, 2,..., EO_j
RS_m	Remaining tool slots on machine m
RT_m	Remaining time on machine m
$T(E_{jm}{}^o)$	Time required for carrying out essential operation O_e of job j on machine m
$S(E_{jm}{}^o)$	Tool slot required for carrying out essential operation O_e of job j on machine m
$T(O_{jm}{}^o)$	Time required for carrying out optional operation O_o of job j on machine m
$S(O_{jm}{}^o)$	Tool slot required for carrying out optional operation O_o of job j on machine m
$B(j, o)$	Set of machines on which operation o of job j can be performed
T_m	Length of scheduling period for the mth machine
P_{jom}	Processing time of operation o of job j on machine m
S_{jom}	Number of tool slots required for processing operation o of job j on machine m

and:

$$X_j = \begin{cases} 1 & \text{if job } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

$$X_{jom} = \begin{cases} 1 & \text{if operation } o \text{ of job } j \text{ is assigned to machine } m \\ 0 & \text{otherwise} \end{cases}$$

Case 1: overloading not permitted The objective of the FMS loading problem is to minimize the total system unbalance, i.e.:

$$\text{minimize}: \sum_{m=1}^{M} T_m - \sum_{m=1}^{M}\sum_{j=1}^{J}\sum_{o=1}^{O_j} B_j P_{jom} X_{jom} \tag{1}$$

The constraints are defined as follows. The overloading of machines is not permitted:

$$\sum_{j=1}^{J}\sum_{o=1}^{O_j} B_j P_{jom} X_{jom} \le T_m \qquad m = 1,\ 2, \ldots,\ M \tag{2}$$

The jobs will be loaded only when there is an availability of tool slots on each machine:

$$\sum_{j=1}^{J}\sum_{o=1}^{O_j} S_{jom} X_{jom} \le S_m \qquad m = 1,\ 2, \ldots,\ M \tag{3}$$

Ensure that a particular operation of a job is done only on one machine:

$$\sum_{G \in B(j,\,o)} X_{joG} \le 1 \qquad j = 1,\ 2, \ldots,\ J \text{ and } o = 1,\ 2, \ldots,\ O_j \tag{4}$$

where G is an element that belongs to $B(j, o)$. Ensure that the job cannot be split:

$$\sum_{o=1}^{O_j}\sum_{m=1}^{M} X_{jom} = X_j O_j \qquad j = 1,\ 2, \ldots,\ J \tag{5}$$

Case 2: overloading permitted The objective of the FMS loading problem is to minimize the absolute value of the total system unbalance:

$$\text{minimize}: \left| \sum_{m=1}^{M} T_m - \sum_{m=1}^{M}\sum_{j=1}^{J}\sum_{o=1}^{O_j} B_j P_{jom} X_{jom} \right| \tag{6}$$

The constraints are defined as follows. The overloading of machines is permitted:

$$\sum_{m=1}^{M}\sum_{j=1}^{J}\sum_{o=1}^{O_j} B_j P_{jom} X_{jom} \le \sum_{m=1}^{M} T_m \qquad m = 1,\ 2, \ldots,\ M \tag{7}$$

The jobs will be loaded only when there is an availability of tool slots on each machine:

$$\sum_{j=1}^{J}\sum_{o=1}^{O_j} S_{jom} X_{jom} \le S_m \qquad m = 1,\ 2, \ldots,\ M \tag{8}$$

Ensure that a particular operation of a job is done only on one machine:

$$\sum_{G \in B(j,\,o)} X_{joG} \le 1 \qquad j = 1,\ 2, \ldots,\ J \text{ and } o = 1,\ 2, \ldots,\ O_j \tag{9}$$

where G is an element that belongs to $B(j, o)$. Ensure that the job cannot be split:

$$\sum_{o=1}^{O_j}\sum_{m=1}^{M} X_{jom} = X_j O_j \qquad j = 1,\ 2, \ldots,\ J \tag{10}$$

If the number of operations is equal for all of the jobs, denoted as $O_1=O_2=O_3=...=O_j=O$, then mathematical formulation in both Case 1 and Case 2 requires $(J \times (M \times O + 1))$ decision variables. But, the number of constraints in Case 1 is $(M + M + J + J \times O)$, whereas Case 2 needs $(1 + M + J + J \times O)$ constraints. Let us consider a problem with six jobs (J=6) having two operations for each (O_j=2), and the jobs are to be loaded on four machines (M=4). Then, the number of decision variables is 54 and the number of constraints is 26 for Case 1. Similarly, for Case 2, the number of decision variables is 54 and the number of constraints is 23. As the problem size increases, the number of variables and constraints will increase further, making it difficult to solve. Hence, there is a need to find a better search technique which can provide a near-optimal solution. In this paper, an attempt has been made to solve the machine-loading problem of an FMS by using a meta-heuristic approach based on PSO.

4 Particle swarm optimization

The particle swarm optimization (PSO) algorithm, originally introduced by Kennedy and Eberhart in 1995 [29], is one of the latest evolutionary optimization techniques inspired by nature. It is based on the metaphor of social interaction and communication of bird flocking and fish schooling. In PSO, the members of the entire population are maintained through the search procedure, so that information is socially shared among individuals to direct the search towards the best position in the search space. In PSO, each member is called a particle, and each particle moves around in the multidimensional search space with a velocity which is constantly updated by the particle's own experience and the experience of the particle's neighbors or the experience of the whole swarm. Generally, PSO is characterized as a simple heuristic of well-balanced mechanism, with the flexibility to enhance and adapt to both global and local exploration abilities. Compared with GAs, PSO has some attractive characteristics. It has memory, which enables it to retain knowledge of good solutions by all particles, whereas previous knowledge of the problem is destroyed once the population changes in GAs. In PSO, there is a mechanism of constructive cooperation and information-sharing between particles. Due to the simple concept, easy implementation, and quick convergence, PSO has gained much attention and has been successfully applied to a wide range of applications, such as the scheduling of an FMS, power and voltage control, neural network training, mass-spring systems, task assignment, supplier selection and ordering problem, automated drilling, state estimation for electric power distribution systems, etc. [30–33]. The application of the PSO algorithm requires that parameters are initialized and the initial population to be generated randomly. After evaluation, the PSO algorithm repeats the following steps iteratively:

- Each particle with its position, velocity, and fitness value updates its personal best (best value of each individual so far) if an improved fitness value is found
- The best particle in the whole swarm with its position and fitness value is, on the other hand, used to update the global best (best particle in the whole swarm)
- Then, each particle updates its velocity based on the experiences of its personal best and the global best in order to update the position of each particle with the velocity currently updated
- Permutation is determined through the smallest position value (SPV) rule 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, the velocities and positions of each particle are updated using Eqs. 11 and 12, respectively:

$$v_{ij}^{t} = w^{t-1} v_{ij}^{t-1} + c_1 r_1 \left(p_{ij}^{t-1} - x_{ij}^{t-1} \right) + c_2 r_2 \left(g_j^{t-1} - x_{ij}^{t-1} \right) \tag{11}$$

$$x_{ij}^{t} = x_{ij}^{t-1} + v_{ij}^{t} \tag{12}$$

where $v_{ij}^{\ t}$ represents the velocity of particle i at iteration t with respect to the 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 the jth dimension. c_1 and c_2 are positive acceleration parameters, called the cognitive and social parameters, respectively, and r_1 and r_2 are uniform random numbers in the interval (0, 1).

w is known as the inertia weight, which is updated as follows:

$$w^{t} = w^{t-1} \times \alpha \tag{13}$$

Table 1 Solution representation of particle X_i^t in the particle swarm optimization (PSO) algorithm

Dimension, j	1	2	3	4	5	6
$x_{ij}^{\ t}$	0.11	1.48	1.21	0.45	1.08	0.32
$v_{ij}^{\ t}$	3.89	2.94	3.08	−0.87	−0.20	3.16
Job sequence, $\pi_i^{\ t}$	1	6	4	5	3	2

where α is a decrement factor. The parameter w controls the impact of the previous velocities on the current velocity. The termination criterion might be a maximum number of iterations or the maximum CPU time to terminate the search.

4.1 Solution representation

One of the most important issues when designing the PSO algorithm lies on its solution representation. In order to construct a direct relationship between the problem domain and the PSO particles for the FMS loading problem, we present n number of dimensions for n number of jobs. In other words, each dimension represents a typical job. In addition, the particle $X_i^t = [x_{i1}^t, x_{i2}^t, \ldots, x_{in}^t]$ corresponds to the continuous position values for n number of jobs in the loading problem. The particle itself does not present a job permutation. Rather, the job sequence $\pi_i^t = [\pi_{i1}^t, \pi_{i2}^t, \ldots, \pi_{in}^t]$ is determined from position values using the SPV rule. Table 1 illustrates the solution representation of particle X_i^t for the FMS loading problem, together with its corresponding velocity and sequence. According to the SPV rule, the smallest position value is $x_{i1}{}^t$=0.11, so the dimension j=1 representing job 1 is assigned to the first

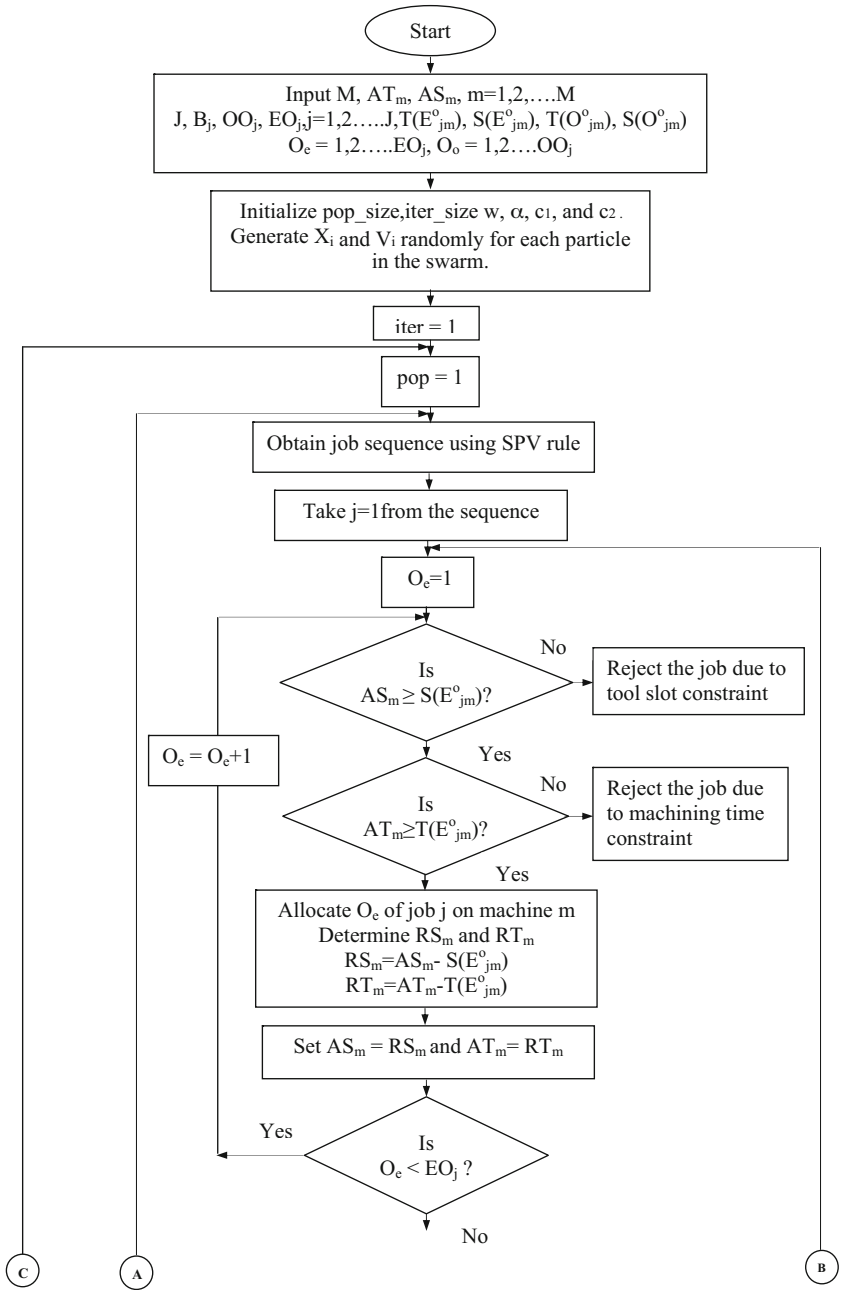
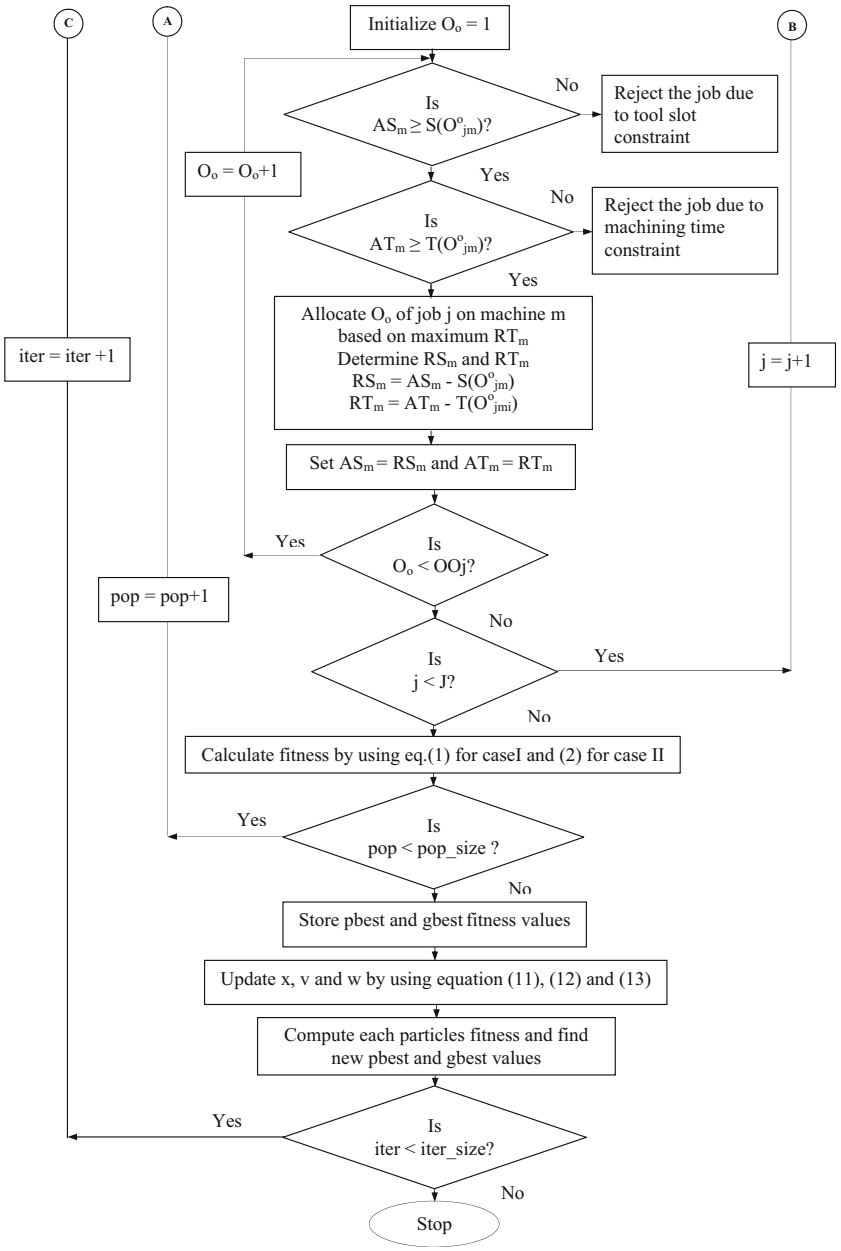


Fig. 1 Flow chart for the particle swarm optimization (PSO) algorithm

Fig. 1 (continued)



position in the job sequence, $\pi_{i1}{}^t=1$ in the sequence; the second smallest position value is $\pi_{i6}{}^t=0.32$, so the dimension $j=6$ representing job 6 is assigned to the second position in the job sequence $\pi_{i6}{}^t=6$, and so on. Finally, the job sequence is obtained as [1, 6, 4, 5, 3, 2], as shown in Table 1. In other words, the dimensions are sorted according to the position values $x_{ij}{}^t$ to construct the sequence.

4.2 The problem of the lack of diversity and the mutation operator

PSO schemes described above typically converge relatively rapidly in the first part of the search and then slow down or stop. This behavior has been attributed to the loss of diversity in the population and a number of researchers have suggested methods to overcome this drawback, with varying degrees of success [34, 35]. Looking at the positions of the particles when the swarm had stagnated, it was clear that the points were very tightly clustered and the velocities were almost zero. The points were often not that far from the global optimum, but the updating equations, due to the almost zero velocity, were unable to generate new solutions, which might lead the swarm out of this state. This behavior can also lead to the whole swarm being trapped in a local optimum, from which it becomes impossible to escape.

Table 2 Description of problem no. 1 (adapted from Mukhopadhyay et al. [14])

Job number	Batch size	Operation number	Unit processing time (min)	Tool slots needed	Machine number
1	8	1	18	1	3
2	9	1	25	1	1, 4
		2	24	1	4
		3	22	1	2
3	13	1	26	2	4, 1
		2	11	3	3
4	6	1	14	1	3
		2	19	1	4
5	9	1	22	2	2, 3
		2	25	1	2
6	10	1	16	1	4
		2	7	1	4, 2, 3
		3	21	1	2, 1
7	12	1	19	1	3, 2, 4
		2	13	1	2, 3, 1
		3	23	3	4
8	13	1	25	1	1, 2, 3
		2	7	1	2, 1
		3	24	3	1

As mutation is capable of introducing diversity in the search procedure, two types mutation have attracted the attention of researchers; mutation of the global best and mutation based on sharing information from neighbors. Because the global best individual attracts all members of the swarm, it is possible to lead the swarm away from a current location by mutating a single individual if the mutated individual becomes the new global best. This mechanism potentially provides a means for both escaping local optima and speeding up the search. 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 a few individuals in the swarm.

In this work, a mutation operator is introduced which mutates the position vectors of a few particles selected randomly. The mutation operation is not executed in every iteration. A function, rand(0, MAXT) is used to return an integer greater than or equal to 0 and less than MAXT. In each iteration, if the elapsed time with no further progress is greater than this random value, the mutation operation is executed. Two reasons may account for this strategy:(1) if there is no premature convergence happening, the execution of PSO will not be affected; (2) the algorithm will not increase computational overheads by very much. The mutation operator randomly swaps two positions of a particle at random. Then, permutations are generated for the new positions. The mutation strategy is depicted in algorithmic form as follows:

```
// DELTA: elapsed time of no further progress //
// MAXT: maximum time of no further progress //
t=0
initialize the swarm
evaluate each particle
while (not-termination-condition) do
t=t+1
update swarm according to Eqs. 11 and 12
if (DELTA>rand(0, MAXT))
do mutation
end if
evaluate the swarm
End
```

Table 3 Comparison of results obtained using the PSO algorithm with other methods (overloading not permitted)

Problem no.	Total number of jobs	Shanker and Srinivasulu [28]		PSO algorithm		Assigned jobs	Rejected jobs
		SU	TP	SU	TP		
1	8	253	39	253	39	{1, 7, 6, 5}	{2, 3, 4, 8}
2	6	388	51	202	63	{6, 1, 3, 5, 4}	{2}
3	5	288	63	72	69	{4, 3, 5, 2}	{1}
4	5	819	51	819	51	{3, 4, 5, 2, 1}	{Φ}
5	6	467	62	219	52	{2, 3, 4, 1}	{5, 6}
6	6	548	51	178	46	{5, 2, 6, 4}	{1, 3}
7	6	189	54	189	54	{4, 3, 2, 1}	{5, 6}
8	7	459	36	448	40	{4, 5, 1, 7}	{2, 3, 6}
9	7	462	79	309	88	{2, 5, 6, 1, 4, 3, 7}	{Φ}
10	6	518	44	184	49	{1, 4, 5, 6}	{2, 3}

Table 4 Summary of results obtained from the PSO algorithm (overloading permitted)

Problem no.	Total number of jobs	PSO algorithm		Assigned jobs	Rejected jobs
		SU	TP		
1	8	152	45	{1, 4, 7, 3, 6}	{8, 2, 5}
2	6	69	50	{5, 2, 1, 4}	{3, 6}
3	5	98	79	{4, 3, 5, 1}	{2}
4	5	819	51	{3, 4, 5, 2, 1}	{Φ}
5	6	9	64	{1, 6, 2, 3, 4}	{5}
6	6	35	62	{5, 2, 1, 3, 4}	{6}
7	6	39	66	{1, 6, 2, 3, 4}	{5}
8	7	101	42	{2, 3, 7, 5, 1}	{4, 6}
9	7	309	88	{1, 7, 6, 2, 3, 4, 5}	{Φ}
10	6	5	58	{5, 4, 1, 6, 3}	{2}

4.3 The proposed algorithm

Step 1 Input the total number of available machines, jobs, batch sizes, tool slots on each machine, operations of all jobs (both essential and optional), and the processing time of every operation of each job.

Step 2 Initialize the parameters such as population size, maximum iteration, decrement factor, inertia weight, and social and cognitive parameters. Generate the initial population randomly. Construct the initial position values of the particle uniformly as $x_{ij}^t = x_{\min} + (x_{\max} - x_{\min}) \times \mathrm{U}(0,\ 1)$, where $x_{\min}$=0.0, $x_{\max}$=4.0, and U(0, 1) is a uniform random number between 0 and 1. Generate the initial velocities of the particle as $v_{ij}^t = v_{\min} + (v_{\max} - v_{\min}) \times \mathrm{U}(0,\ 1)$, where $v_{\min}$=−4.0, $v_{\max}$= 4.0, and U(0, 1) is a uniform random number between 0 and 1.

Step 3 Get the initial sequence by using the SPV rule. Then, select the first job from that sequence and do the following:

(a) First, load the essential operation on the machine if and only if the available machining time is greater than the time required by the essential operation; otherwise, reject the job.

(b) Similarly, load the optional operation if and only if the available machining time and tool slot is greater than the time and tool slot required by the optional operation on the basis of the machine having the maximum available time; otherwise, reject the job.

Step 4 Evaluate each particle's fitness (system unbalance) by using Eq. 1 for Case 1 and Eq. 6 for Case 2, while satisfying their respective constraints.

Step 5 Find out the personal best (pbest) and the global best (gbest).

Step 6 If no progress in the pbest value is observed for an elapsed period of DELTA, carry out the mutation of a particle using the mutation strategy as outlined in Sect. 4.2, provided that DELTA is greater than a random number between 0 and the maximum time of no progress (MAXT).

Step 7 Update the velocity, position, and inertia weight by using Eqs. 11, 12, and 13, respectively.

Step 8 Compute each particle's fitness similar to Step 3 and find the new pbest and gbest.

Step 9 Terminate if the maximum number of iterations is reached and store the gbest value; otherwise, go to Step 2.

5 Results and discussion

The proposed PSO algorithm for the FMS loading problem is coded in Visual C^{++} and implemented on a Pentium IV

Table 5 Comparison of the results obtained using the modified PSO algorithm with other methods (overloading permitted)

Problem no.	Total number of jobs	Srinivas et al. [17]		Swarnkar and Tiwari [22]		Nagarjuna et al. [18]		Modified PSO algorithm	
		SU	TP	SU	TP	SU	TP	SU	TP
1	8	14	48	14	48	14	48	3	49
2	6	500	57	234	63	18	46	69	50
3	5	72	69	128	73	94	69	32	48
4	5	819	51	819	51	819	51	819	51
5	6	467	62	364	76	175	53	9	64
6	6	500	57	69	64	69	64	35	62
7	6	486	63	177	54	165	54	39	66
8	7	72	48	63	48	13	44	52	50
9	7	309	88	309	88	309	88	309	88
10	6	84	54	122	56	82	54	5	58

PC. The algorithm is shown as a flow chart in Fig. 1. The performance of the PSO algorithm is evaluated by using ten benchmark problems available in the open literature. The description of problem no. 1 having eight jobs is shown in Table 2. Since different methods have been used by researchers for the calculation of system unbalance, the machine-loading problem of an FMS has been solved in this paper by considering two cases to examine the robustness of the PSO algorithm. Computational results for Case 1 and Case 2 are summarized in Tables 3 and 4, respectively, using standard PSO. It is evident from Table 3 that the PSO algorithm outperforms the other solution methodologies for all instances. For example, the PSO algorithm yields a system unbalance of 202 and a throughput of 63, whereas Shanker and Srinivasulu [28] arrived at a system unbalance of 388 and a throughput of 51 for problem no. 2. In the case that overloading on machines is permitted, the PSO algorithm is capable of either outperforming existing methods or at least producing an equal solution, as indicated in Table 4. However, the computational effort in both cases is much less compared to other solution methods. It was observed that, for the test problems selected, around 50 iterations were required for obtaining a global/near-optimal solution [21], but the PSO algorithm could produce best solutions within 20 iterations and, hence, it requires less computational effort.

One of the major drawbacks of PSO is its premature convergence. In order to eliminate this and to improve the solution quality, the mutation operator is adopted from GAs. The results of the proposed modified PSO are given in Table 5. The results indicate that PSO with mutation provides a significant improvement in performance. For example, the modified PSO algorithm yields a system unbalance of 3 and a throughput of 49 for problem no. 1, whereas Nagarjuna et al. [18] arrived at a system unbalance of 14 and a throughput of 48. For the same problem, Srinivas et al. [17] and Swarnkar and Tiwari [22] also reported a system unbalance of 14 and a throughput of 48. However, the improvement in PSO is obtained with marginal computational efforts compared to the other methods. Figure 2 illustrates the convergence behavior of

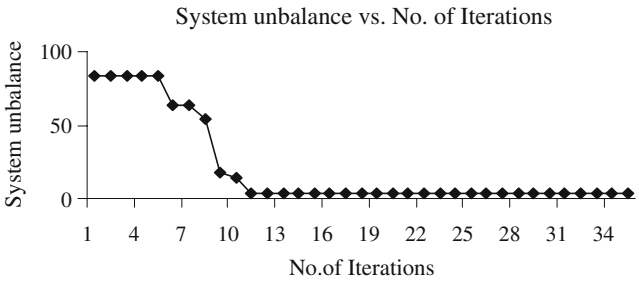


Fig. 2 Convergence curve for PSO

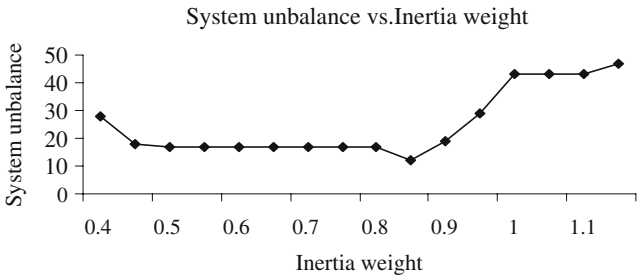


Fig. 3 Effect of inertia weight on system unbalance

the PSO algorithm for the different parameter combinations. It can be observed from the figure that the algorithm can achieve the optimal solution after 12 iterations for problem no. 1, when population size is fixed at 25, w=0.85, α=0.9, and c_1=c_2=2 because no further improvement is observed beyond 12 iterations.

The study not only aims at providing a PSO algorithm that produces the best possible solutions to a set of optimization problems, but it also illustrates the importance of the correct selection of the parameters, such as inertia weight and population size. An important parameter in PSO is the inertia weight w, which controls the impact of previous velocities on the current velocity. In order to investigate the effect of w on the solution quality, the value of w is varied from 0.4 to 1.15 in increments of 0.05, keeping all other parameters constant. From Fig. 3, it can be observed that the best fitness value (system unbalance) is obtained at w=0.85 when population size is 25 for problem no. 1. The value of w should not be very low or very high, but its best value must be experimentally discovered, depending on the type of problem. Therefore, it can be stated that a low value of w leads to small memory retention capacity of the particles and it takes a long time to achieve a global optimum. However, a very large value of w causes excessive memory retention of the particles and makes it difficult to escape from a local optimum. Another important parameter in PSO is the population size. To study the effect of population size on the solution quality, it is increased from 5 to 50 in increments of 5 for the test problems, keeping all other parameters constant. Figure 4 shows the effect of the population on system unbalance. It can be

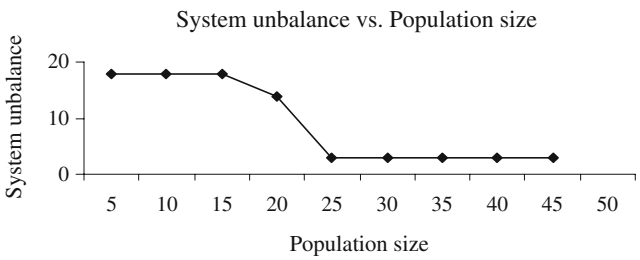


Fig. 4 Best results obtained for different population sizes

observed that, as the population size increases, the system unbalance decreases to a certain extent and further increases of the population has no effect on the solution quality. This may be attributed to the fact that diversity in the solution space increases as the population size increases, but large increases in the population size causes a random or worst point of search and increases the possibility of trapping at a local optimum. However, the population size must be maintained at least two times the number of jobs in order to obtain an improved solution.

6 Conclusions

This paper presents an efficient and reliable evolutionary-based approach to solve the flexible manufacturing system (FMS) loading problem. The proposed approach utilizes the global and local exploration capabilities of particle swarm optimization (PSO) to search for the optimal solution by taking into account the availability of tool slots and machining time constraints. The key advantage of PSO is its computational efficiency and a smaller number of parameters are required to be adjusted in order to obtain the optimum result compared to related techniques. Extensive computational experiments have been conducted on different benchmark problems to show the effectiveness of the proposed approach. A comparative study has been carried out for the same problem with similar objective functions and constraints, and the computational experience manifests that the proposed meta-heuristic approach based on PSO outperforms the existing methodologies as far as solution quality is concerned with reasonable computational efforts. Many researchers have developed heuristics to minimize the system unbalance, but at the expense of throughput, which can be visualized in Tables 2 and 4.

Although the objective of this study is to minimize system unbalance, the proposed meta-heuristic based on PSO with mutation not only minimizes system unbalance, but it also simultaneously increases the throughput for most of the instances. A modified PSO algorithm is proposed in this paper to avoid premature convergence with the introduction of the mutation operation. The performance of this algorithm is compared to the standard PSO algorithm and other related techniques. The result obtained by the modified PSO scheme is promising and encouraging. It is evident from this study that the overloading of machines is a viable proposition for minimizing the system unbalance, but it involves cost. Therefore, a tradeoff between the balancing of loads on machines and cost incurred must be made.

In future, the study can be extended to solve the loading problem using the same meta-heuristic approach in a multiple-objective framework, i.e., combined objective of the minimization of the system unbalance and the maximization of throughput. Consideration of multiple-objective PSO based on Pareto dominance rough sets or fuzzy sets to balance conflicting technological, economic, and environmental targets may give a new direction for research in the field of FMS. More realistic variables and constraints, such as the availability of pallets, jigs, fixtures, automatic guided vehicles (AGVs), etc., in addition to tool slots and machining time may be included to solve loading problems in future studies.

Acknowledgment We extend our hearty thanks to the anonymous reviewer for their constructive suggestions that helped us to improve the quality of the technical and literal content of the manuscript.

References

1. Stecke KE (1983) Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems. Manage Sci 29(3):273–288
2. Stecke KE, Solberg JJ (1981) Loading and control policies for a flexible manufacturing system. Int J Prod Res 19(5):481–490
3. Berrada M, Stecke KE (1986) A branch and bound approach for machine load balancing in flexible manufacturing systems. Manage Sci 32(10):1316–1335
4. Sarin SC, Chen CS (1987) The machine loading and tool allocation problem in a flexible manufacturing system. Int J Prod Res 25(7):1081–1094
5. Chen IJ, Chung C-H (1991) Effects of loading and routing decisions on performance of flexible manufacturing systems. Int J Prod Res 29(11):2209–2225
6. Atmani A, Lashkari RS (1998) A model of machine-tool selection and operation allocation in FMS. Int J Prod Res 36(5):1339–1349
7. Gamila MA, Motavalli S (2003) A Modeling technique for loading and scheduling problems in FMS. Robot Comput-Int Manuf 19(1–2):45–54
8. Shanker K, Tzen Y-JJ (1985) A loading and dispatching problem in a random flexible manufacturing system. Int J Prod Res 23 (3):579–595
9. O'Grady PJ, Menon U (1987) Loading a flexible manufacturing system. Int J Prod Res 25(7):1053–1068
10. Liang M, Dutta SP (1992) Combined part selection, load sharing and machine loading problem in hybrid manufacturing systems. Int J Prod Res 30(10):2335–2349
11. Liang M, Dutta SP (1993) Solving a combined part-selection, machine-loading, and tool-configuration problem in flexible manufacturing systems. Prod Oper Manag 2:97–113
12. Liang M, Dutta SP (1993) An integrated approach to the part selection and machine loading problem in a class of flexible manufacturing systems. Eur J Oper Res 67:387–404
13. Stecke KE, Tallbot FB (1983) Heuristic algorithms for flexible manufacturing systems. In: Proceedings of the 7th International Conference on Production Research, Windsor, Ontario, Canada, August 1983, pp 570–576
14. Mukhopadhyay SK, Midha S, Krishna VM (1992) A heuristic procedure for loading problems in flexible manufacturing systems. Int J Prod Res 30(9):2213–2228
15. Tiwari MK, Hazarika B, Vidyarthi NK, Jaggi P, Mukhopadhyay SK (1997) A heuristic solution approach to the machine loading problem of an FMS and its Petri net model. Int J Prod Res 35 (8):2269–2284

16. Moreno AA, Ding F-Y (1993) Heuristics for the FMS-loading and part-type-selection problems. Int J Flex Manuf Syst 5:287–300
17. Srinivas A, Tiwari MK, Allada V (2004) Solving the machine-loading problem in a flexible manufacturing system using a combinatorial auction-based approach. Int J Prod Res 42(9): 1879–1893
18. Nagarjuna N, Mahesh O, Rajagopal K (2006) A heuristic based on multi-stage programming approach for machine-loading problem in a flexible manufacturing system. Robot Comput-Int Manuf 22:342–352
19. Mukhopadhyay SK, Singh MK, Srivastava R (1998) FMS machine loading: a simulated annealing approach. Int J Prod Res 36(6):1529–1547
20. Kumar N, Shanker K (2000) A genetic algorithm for FMS part type selection and machine loading. Int J Prod Res 38(16): 3861–3887
21. Tiwari MK, Vidyarthi NK (2000) Solving machine loading problems in a flexible manufacturing system using a genetic algorithm based heuristic approach. Int J Prod Res 38(14): 3357–3384
22. Swarnkar R, Tiwari MK (2004) Modeling machine loading problem of FMSs and its solution methodology using a hybrid tabu search and simulated annealing-based heuristic approach. Robot Comput-Int Manuf 20(3):199–209
23. Vidyarthi NK, Tiwari MK (2001) Machine loading problem of FMS: a fuzzy-based heuristic approach. Int J Prod Res 39(5): 953–979
24. Chan FTS, Swarnkar R (2006) Ant colony optimization approach to a fuzzy goal programming model for a machine tool selection and operation allocation problem in an FMS. Robot Comput-Int Manuf 22(4):353–362
25. Gupta JND, Luong LHS, Nguyen VH (1999) Part dispatching and machine loading in flexible manufacturing systems using central queues. Int J Prod Res 37(6):1427–1435
26. Kost GG, Zdanowicz R (2005) Modeling of manufacturing systems and robot motions. J Mater Process Technol 164–165:1369–1378
27. Zolfaghari S, Liang M (1999) Jointly solving the group scheduling and machining speed selection problems: a hybrid tabu search and simulated annealing approach. Int J Prod Res 37 (10):2377–2397
28. Shanker K, Srinivasulu A (1989) Some solution methodologies for loading problems in a flexible manufacturing system. Int J Prod Res 27(6):1019–1034
29. Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks (ICNN'95), Perth, Australia, November/December 1995, vol 4, pp 1942–1948
30. Jerald J, Asokan P, Prabaharan G, Saravanan R (2005) Scheduling optimisation of flexible manufacturing systems using particle swarm optimisation algorithm. Int J Adv Manuf Technol 25:964–971
31. van den Bergh F, Engelbecht AP (2000) Cooperative learning in neural networks using particle swarm optimizers. S Afr Comput J 26:84–90
32. Salman A, Ahmad I, Al-Madani S (2002) Particle swarm optimization for task assignment problem. Microprocess Microsyst 26:363–371
33. Abido MA (2002) Optimal power flow using particle swarm optimization. Electr Power Energy Syst 24:563–571
34. Riget J, Vesterstroem JS (2002) A diversity guided particle swarm optimizer—the ARPSO. Department of Computer Science, University of Aarhus, Denmark, technical report no 2002-02 EVA Life
35. Krink T, Lovhjerg M (2002) The lifecycle model: combining particle swarm optimisation, genetic algorithms and hillclimbers. In: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature PPSN 2002), Granada, Spain, September 2002, pp 621–630