Int J Adv Manuf Technol (2009) 45:749–758
DOI 10.1007/s00170-009-1970-6

ORIGINAL ARTICLE

A particle swarm optimization for multi-objective flowshop scheduling

D. Y. Sha · Hsing-Hung Lin

Received: 5 September 2008 / Accepted: 6 February 2009 / Published online: 24 February 2009

Abstract The academic approach of single-objective flowshop scheduling has been extended to multiple objectives to meet the requirements of realistic manufacturing systems. Many algorithms have been developed to search for optimal or near-optimal solutions due to the computational cost of determining exact solutions. This paper provides a particle swarm optimization-based multi-objective algorithm for flowshop scheduling. The proposed evolutionary algorithm searches the Pareto optimal solution for objectives by considering the makespan, mean flow time, and machine idle time. The algorithm was tested on benchmark problems to evaluate its performance. The results show that the modified particle swarm optimization algorithm performed better in terms of searching quality and efficiency than other traditional heuristics.

Keywords PSO · Multi-objective · Flowshop scheduling · Pareto optimal

1 Introduction

Production scheduling in real environments has become a significant challenge in enterprises maintaining their competitive positions in rapidly changing markets. Flowshop scheduling problems have attracted much attention in academic circles in the last five decades since Johnson's initial research. Most of these studies have focused on finding the exact optimal solution. A brief overview of the evolution of flowshop scheduling problems and possible approaches to their solution over the last 50 years has been provided by Gupta and Stafford [5]. That survey indicated that most research on flowshop scheduling has focused on single-objective problems, such as minimizing completion time, total flow time, or total tardiness. Numerous heuristic techniques have been developed for obtaining the approximate optimal solution to NP-hard scheduling problems. A complete survey of flowshop scheduling problems with makespan criterion and contributions, including exact methods, constructive heuristics, improved heuristics, and evolutionary approaches from 1954 to 2004, was offered by Hejazi et al. [7]. Ruiz et al. [24] also presented a review and comparative evaluation of heuristics and meta-heuristics for permutation flowshop problems with the makespan criterion. The NEH algorithm [17] has been shown to be the best constructive heuristic for Taillard's benchmarks [28] while the iterated local search [27] method and the genetic algorithm (GA) [23] are better than other meta-heuristic algorithms.

Most studies of flowshop scheduling have focused on a single objective that could be optimized independently. However, empirical scheduling decisions might not only involve the consideration of more than one objective, but also require minimizing the conflict between two or more objectives. In addition, finding the exact solution to scheduling problems is computationally expensive because such problems are NP-hard. Solving a scheduling problem with multiple objectives is even more complicated than solving a single-objective problem. Approaches including meta-heuristics and memetics have been developed to reduce the complexity and improve the efficiency of solutions.

D. Y. Sha
Department of Industrial Engineering and System Management, Chung Hua University,
Hsinchu, Taiwan, Republic of China

H.-H. Lin (✉)
Department of Industrial Engineering and Management, National Chiao Tung University,
Hsinchu, Taiwan, Republic of China
e-mail: hsinhung@gmail.com

Hybrid heuristics combining the features of different methods in a complementary fashion have been a hot issue in the fields of computer science and operational research [15]. Ponnambalam et al. [19] considered a weighted sum of multiple objectives, including minimizing the makespan, mean flow time, and machine idle time as a performance measurement, and proposed a multi-objective algorithm using a traveling salesman algorithm and the GA for the flowshop scheduling problem. Rajendran et al. [21] approached the problem of scheduling in permutation flowshop using two ant colony optimization (ACO) approaches, first to minimize the makespan, and then to minimize the sum of the total flow time. Yagmahan [30] was the first to apply ACO meta-heuristics to flowshop scheduling with the multiple objectives of makespan, total flow time, and total machine idle time.

The literature on multi-objective flowshop scheduling problems can divided into two groups: a priori approaches with assigned weights of each objective, and a posteriori approaches involving a set of non-dominated solutions [18]. There is also a multi-objective GA (MOGA) called PGA-ALS, designed to search non-dominated sequences with the objectives of minimizing makespan and total flow time. The multi-objective solutions are called non-dominated solutions (or Pareto optimal solutions in the case of Pareto optimality). Eren et al. [4] tackled a multi-criteria two-machine flowshop scheduling problem with minimization of the weighted sum of total completion time, total tardiness, and makespan.

Particle swarm optimization (PSO) is an evolutionary technique for unconstrained continuous optimization problems proposed by Kennedy et al. [10] The PSO concept is based on observations of the social behavior of animals such as birds in flocks, fish in schools, and swarm theory. To minimize the objective of maximum completion time (i.e., the makespan), Liu et al. [15] invented an effective PSO-based memetic algorithm for the permutation flowshop scheduling problem. Jarboui et al. [9] developed a PSO algorithm for solving the permutation flowshop scheduling problem; this was an improved procedure based on simulated annealing. PSO was recommended by Tasgetiren et al. [29] to solve the permutation flowshop scheduling problem with the objectives of minimizing makespan and the total flow time of jobs. Rahimi-Vahed et al. [22] tackled a bi-criteria permutation flowshop scheduling problem where the weighted mean completion time and the weighted mean tardiness were minimized simultaneously. They exploited a new concept called the *ideal point* and a new approach to specifying the superior particle's position vector in the swarm that is designed and used for finding the locally Pareto optimal frontier of the problem. Due to the discrete nature of the flowshop scheduling problem, Lian et al. [14] addressed permutation flowshop scheduling with a minimized makespan using a novel PSO. All these approaches have demonstrated the advantages of the PSO method: simple structure, immediate applicability to practical problems, ease of implementation, quick solution, and robustness.

The aim of this paper is to explore the development of PSO for elaborate multi-objective flowshop scheduling problems. The original PSO was used to solve continuous optimization problems. Due to the discrete solution spaces of scheduling optimization problems, we modified the particle position representation, particle movement, and particle velocity in this study.

The remainder of this paper is organized as follows. Section 2 contains a formulation of the flowshop scheduling problem with two objectives. Section 3 describes the algorithm of the proposed PSO approach. Section 4 contains the simulated results of benchmark problems. Section 5 provides some conclusions and future directions.

2 Problem formulation

The problem of scheduling in flowshops has been the subject of much investigation. The primary elements of flowshop scheduling include a set of m machines and a collection of n jobs to be scheduled on the set of machines. Each job follows the same process of machines and passes through each machine only once. Each job can be processed on one and only one machine at a time, whereas each machine can process only one job at a time. The processing time of each job on each machine is fixed and known in advance. We formulate the multi-objective flowshop scheduling problem using the following notation:

- n is the total number of jobs to be scheduled,
- m is the total number of machines in the process,
- $t(i, j)$ is the processing time for job i on machine j (i=1, 2,…n) and (j=1,2,…m), and
- $\{\pi_1, \pi_2, \ldots, \pi_n\}$ is the permutation of jobs.

The objectives considered in this paper can be calculated as follows:

- Completion time (makespan) $C(\pi, j)$:

$$\begin{aligned} &C(\pi_1, 1) = t(\pi_1, 1)\\ &C(\pi_i, 1) = C(\pi_{i-1}, 1) + t(\pi_i, 1) i = 2, \ldots, n\\ &C(\pi_1, j) = C(\pi_1, j-1) + t(\pi, j) j = 2, \ldots, m\\ &C(\pi_i, j) = \max\{C(\pi_{i-1}, j), C(\pi_i, j-1)\} + t(\pi_i, j)\\ &\qquad i = 2, \ldots, n;\ j = 2, \ldots, m \end{aligned}$$

- Makespan, $f_{C\max} = C(\pi_n, m)$,
- Mean flow time, $f_{\mathrm{MFT}} = \left[\sum_{i=1}^{n} C(\pi_i, m)\right] \Big/ n$,

- Machine idle time, and
- $f_{MIT} = \{C(\pi_1, j-1) + \sum_{i=2}^{n} \{\max\{C(\pi_i, j-1) - C(\pi_{i-1}, j), 0\}\} | j = 2...m\}$

3 Basic PSO concept

PSO is an evolutionary technique (Kennedy et al. [10]) for solving unconstrained continuous optimization problems. The PSO concept is based on observations of the social behavior of animals. The population consisting of individuals (particles) is assigned a randomized initial velocity according each individual's own movement experience and that of the rest of the population. The relationship between the swarm and the particles in PSO is similar to the relationship between the population and the chromosomes in the GA.

The PSO problem solution space is formulated as a search space. Each position of the particles in the search space is a correlated solution of the problem. Particles cooperate to determine the best position (solution) in the search space (solution space).

Suppose that the search space is D-dimensional and there are m particles in the swarm. Each particle is located at position $X_i=\{x_{i1}, x_{i2}, \ldots, x_{iD}\}$ with velocity $V_i=\{v_{i1}, v_{i2}, \ldots, v_{iD}\}$, where $i=1, 2, \ldots, m$. In the PSO algorithm, each particle moves toward its own best position (pbest) denoted as Pbest$_i$={pbest$_{i1}$, pbest$_{i2}$,..., pbest$_{in}$}. The best position of the whole swarm (gbest) denoted as Gbest={gbest$_1$, gbest$_2$, ..., gbest$_n$} with each iteration. Each particle changes its position according to its velocity, which is randomly generated toward the pbest and gbest positions. For each particle r and dimension s, the new velocity v_{rs} and position x_{rs} of particles can be calculated by the following equations:

$$v_{rs}^{t} = w \times v_{rs}^{t-1} + c_1 \times rand_1 \times \left(pbest_{rs}^{t-1} - x_{rs}^{t-1}\right) + c_2 \times rand_2 \times \left(gbest_{s}^{t-1} - x_{rs}^{t-1}\right) \quad (1)$$

$$x_{rs}^{t} = x_{rs}^{t-1} + v_{rs}^{t-1} \quad (2)$$

where t *is* the iteration number. The inertial weight w is used to control exploration and exploitation. A large value of w keeps particles at high velocity and prevents them from becoming trapped in local optima. A small value of w maintains particles at low velocity and encourages them to exploit the same search area. The constants c_1 and c_2 are acceleration coefficients that determine whether particles prefer to move closer to the pbest or gbest positions. The $rand_1$ and $rand_2$ are independent random numbers uniformly distributed between 0 and 1. The termination criterion of the PSO algorithm includes the maximum number of generations, the designated value of pbest, and no further improvement in pbest. The standard PSO process outline is as follows.

Step 1: initialize a population of particles with random positions and velocities on D dimensions in the search space.

Step 2: update the velocity of each particle according to Eq. (1).

Step 3: update the position of each particle according to Eq. (2).

Step 4: map the position of each particle into the solution space and evaluate its fitness value according to the desired optimization fitness function. Simultaneously update the pbest and gbest positions if necessary.

Step 5: loop to Step 2 until an exit criterion is met, usually a sufficient goodness of fitness or a maximum number of iterations.

The original PSO was designed for a continuous solution space. We modified the PSO position representation, particle velocity, and particle movement as described in the next section to make PSO suitable for combinational optimization problems.

4 Formation of the proposed PSO

There are two different representations of particle position associated with a schedule. Zhang [31] demonstrated that permutation-based position representation outperforms priority-based representation. While we have chosen to implement permutation-based position representation, we must also adjust the particle velocity and particle movement as described in Sections 4.2 and 4.3. We have also included the maintenance of Pareto optima and local search procedures to achieve better performance.

4.1 Position representation

In this study, we randomly generated a group of particles (solutions) represented by a permutation sequence that is an ordered list of operations. The following example is a permutation sequence for a six-job permutation flowshop scheduling problem, where j_n is the operation of job n.

Index :	1	2	3	4	5	6
Permutation :	j_4	j_3	j_1	j_6	j_2	j_5

An operation earlier in the list has a higher priority of being placed into the schedule. We used a list with a length of n for an n-job problem in our algorithm to represent the position of particle k, i.e.,

$X^k = \left[x_1^k x_2^k \dots x_n^k\right]$,
x_i^k is the priority of j_i in particle k.

Then, we convert the permutation list to a priority list. The x_i^k is a value randomly initialized to some value between (p–0.5) and (p + 0.5). This means $x_i^k \leftarrow p + \text{rand} - 0.5$, where p is the location (index) of j_i in the permutation list, and *rand* is a random number between 0 and 1. Consequently, the operation with smaller x_i^k has a higher priority for scheduling. The permutation list mentioned above can be converted to

$X^k = [2.7\ \ 5.2\ \ 1.8\ \ 0.6\ \ 6.3\ \ 3.9]$

4.2 Particle velocity

The original PSO velocity concept is that each particle moves according to the velocity determined by the distance between the previous position of the particle and the gbest (pbest) solution. The two major purposes of the particle velocity are to move the particle toward the gbest and pbest solutions, and to maintain the inertia to prevent particles from becoming trapped in local optima.

In the proposed PSO, we concentrated on preventing particles from becoming trapped in local optima rather than moving them toward the gbest (pbest) solution. If the priority value increases or decreases with the present velocity in this iteration, we maintain the priority value increasing or decreasing at the beginning of the next iteration with probability *w*, which is the PSO inertial weight. The larger the value of *w* is, the greater the number of iterations over which the priority value keeps increasing or decreasing, and the greater the difficulty the particle has returning to the current position. For an n-job problem, the velocity of particle k can be represented as

$V^k = \left[v_1^k\ v_2^k \dots v_n^k\right], v_i^k \in \{-1, 0, 1\}$
where v_i^k is the velocity of j_i of particle k.

The initial particle velocities are generated randomly. Instead of considering the distance from x_i^k to $\text{pbest}_i^k(\text{gbest}_i)$, our PSO considers whether the value of x_i^k is larger or smaller than $\text{pbest}_i^k(\text{gbest}_i)$ If x_i^k has decreased in the present iteration, this means that $\text{pbest}_i^k(\text{gbest}_i)$ is smaller than x_i^k, and x_i^k is set moving toward $\text{pbest}_i^k(\text{gbest}_i)$ by letting $v_i^k \leftarrow -1$. Therefore, in the next iteration, x_i^k is kept decreasing by one (i.e., $x_i^k \leftarrow x_i^k - 1$) with probability *w*. Conversely, if x_i^k has increased in this iteration, this means that $\text{pbest}_i^k(\text{gbest}_i)$ is larger than x_i^k, and x_i^k is set moving toward $\text{pbest}_i^k(\text{gbest}_i)$ by letting $v_i^k \leftarrow 1$. Therefore, in the next iteration, x_i^k is kept increasing by one (i.e. $x_i^k \leftarrow x_i^k + 1$) with probability *w*.

The inertial weight *w* influences the velocity of particles in PSO. We randomly update velocities at the beginning of each iteration. For each particle k and operation j_i, if v_i^k is not equal to 0, v_i^k is set to 0 with probability (1−*w*). This ensures that x_i^k stops increasing or decreasing continuously in this iteration with probability (1−*w*).

4.3 Particle movement

The particle movement is based on the insertion operator proposed by Sha et al. [25, 26]. The insertion operator is introduced to the priority list to reduce computational complexity. We illustrate the effect of the insertion operator using the permutation list example described above. If we wish to insert j_4 into the third location of the permutation list, we must move j_6 to the sixth location, move j_1 to the fifth location, move j_2 to the fourth location, and then insert j_4 in the third location. The insertion operation comprising these actions costs O(n/2) on average. However, the insertion operator used in this study need only set $x_i^k \leftarrow 3 + \text{rand} - 0.5$ when we want to insert j_5 in the third location of the permutation. This requires only one step for each insertion. If the random number rand equals 0.1, for example, after j_4 is inserted into the third location, then X^k becomes $X^k = [2.7\ 5.2\ 1.8\ 0.6\ 2.6\ 3.9]$.

If we wish to insert j_i into the pth location in the permutation list, we could set $x_i^k \leftarrow p + \text{rand} - 0.5$. The location of operation j_i in the permutation sequence of the kth pbest and gbest solutions are pbest_i^k and gbest_i, respectively. As particle k moves, if v_i^k equals 0 for all j_i, then x_i^k is set to $\text{pbest}_i^k + \text{rand} - 0.5$ with probability c_1 and set to gbest_i + rand − 0.5 with probability c_2, where rand is a random number between 0 and 1, c_1 and c_2 are constants between 0 and 1, and $c_1 + c_2 \leq 1$. We explain this concept by assuming specific values for V^k, X^k, pbest^k, gbest, c_1, and c_2.

$V^k = [-1\ 0\ 0\ 1\ 0\ 0]$,
$X^k = [2.7\ 5.2\ 1.8\ 0.6\ 6.3\ 3.9]$,
$\text{pbest}^k = [5\ 1\ 4\ 6\ 3\ 2]$,
$\text{gbest} = [6\ 3\ 4\ 5\ 1\ 2], \text{c}_1 = 0.8, c_2 = 0.1.$

- For j_1, since $v_1^k \neq 0$ and $x_1^k \leftarrow x_1^k + v_1^k$, then $x_1^k = 1.7$.
- For j_2, since $v_2^k = 0$, the generated random number rand_1=0.6. Since $\text{rand}_1 \leq c_1$, then the generated random number rand_2=0.3. Since $\text{pbest}_2^k \leq x_2^k$, set $v_2^k \leftarrow -1$ and $x_2^k \leftarrow \text{pbest}_2^k + \text{rand}_2 - 0.5$, i.e., $x_2^k = 0.8$.
- For j_3, since $v_3^k = 0$, the generated random number rand_1=0.93. Since $\text{rand}_1 > c_1 + c_2$, x_3^k and v_3^k do not need to be changed.

- For j_4, since $v_4^k = 1$, then $x_4^k \leftarrow x_4^k + v_4^k$, i.e., $x_4^k = 1.6$.
- For j_5, since $v_5^k = 0$, the generated random number rand_1=0.85. Since $c_1 < \text{rand}_1 \leq c_1 + c_2$, the generated random number rand_2=0.7. Since $\text{gbest}_5 \leq x_5^k$, set $v_5^k \leftarrow -1$. Then $x_5^k \leftarrow \text{gbest}_5 + \text{rand}_2 - 0.5$, i.e., $x_5^k = 1.2$.
- For j_6, since $v_6^k = 0$, the generated random number $rand_1$=0.95. Since $\text{rand}_1 > c_1 + c_2$, x_6^k and v_6^k do not need to be changed.

Therefore, after particle k moves, the V^k and X^k are

$$V^k = [-1 \quad -1 \quad 0 \quad 1 \quad -1 \quad 0]$$
$$X^k = [1.6 \quad 0.8 \quad 1.8 \quad 1.7 \quad 1.2 \quad 3.9]$$

In addition, we use a mutation operator in our PSO algorithm. After moving a particle to a new position, we randomly choose an operation and then mutate its priority value x_i^k in accordance with v_i^k. If $x_i^k \leq (n/2)$, we randomly set x_i^k to a value between $(n/2)$ and n, and set $v_i^k \leftarrow 1$. If $x_i^k > (n/2)$, we randomly set x_i^k to a value between 0 and $(n/2)$, and set $v_i^k \leftarrow -1$.

4.4 Pareto optimal set maintenance

Real empirical scheduling decisions often involve not only the consideration of more than one objective at a time, but also must prevent the conflict of two or more objectives. The solution set of the multi-objective optimization problem with conflicting objective functions consistent with the solutions so that no other solution is better than all other objective functions is called *Pareto optimal*. A multi-objective minimization problem with m decision variables and n objectives is given below to describe the concept of Pareto optimality.

$$\begin{aligned}&\text{Minimize } F(x) = (f_1(x), f_2(x), \ldots, f_n(x))\\&\text{where}, x \in \Re^m, F(x) \in \Re^n\end{aligned}$$

A solution p is said to dominate solution q if and only if

$$\begin{aligned}&f_k(p) \leq f_k(q) \quad \forall k \in \{1, 2, \ldots, n\}\\&f_k(p) < f_k(q) \quad \exists k \in \{1, 2, \ldots, n\}\end{aligned}$$

Non-dominated solutions are defined as solutions that dominate the others but do not dominate themselves. Solution p is said to be a Pareto optimal solution if there exists no other solution q in the feasible space that could dominate p. The set including all Pareto optimal solutions is referred to as the Pareto optimal or Pareto optimalPareto optimal set. A graph plotted using collected Pareto optimal solutions in feasible space is referred to as the *Pareto front*.

The external Pareto optimal set is used to produce a limited size of non-dominated solutions (Knowles et al., [11]; Zitzler et al. [32]). The maximum size of the archive set is specified in advance. This method is used to avoid missing fragments of the non-dominated front during the search process. The Pareto optimal front is formed as the archive is updated iteratively. When the archive set is sufficiently empty and a new non-dominated solution is detected, the new solution enters the archive set. As the new solution enters the archive set, any solution already there that is dominated by this solution will be removed. When the maximum archive size reaches its preset value, the archive set must decide which solution should be replaced. In this study, we propose a novel Pareto archive set update process to preclude losing non-dominated solutions when the Pareto archive set is full. When a new non-dominated solution is discovered, the archive set is updated when one of the following situations occurs: either the number of solutions in the archive set is less than the maximum value, or if the number of solutions in the archive set is equal to or greater than the maximum value, then the one solution in the archive set that is most dissimilar to the new solution is replaced by the new solution. We measure the dissimilarity by the Euclidean distance. A longer distance implies a higher dissimilarity. The non-dominated solution in the Pareto archive set with the longest distance to the newly found solution is replaced. For example, the distance (d_{ij}) between X^1 and X^2 is calculated as

$$X^1 = [2.7\ 5.2\ 1.8\ 0.6\ 6.3\ 3.9]$$
$$X^2 = [1.6\ 0.8\ 1.8\ 1.7\ 1.2\ 3.9]$$
$$d_{ij} = \sqrt{(2.7-1.6)^2 + (5.2-0.8)^2 + (0.6-1.7)^2 + (6.3-1.2)^2} = 6.91$$

The Pareto archive set is updated at the end of each iteration in the proposed PSO.

4.5 Diversification strategy

If all the particles have the same non-dominated solutions, they will be trapped in the local optimal. To prevent this, a diversification strategy is proposed to keep the non-dominated solutions different. Once any new solution is generated by the particles, the non-dominated solution set is updated according to one of three situations.

1. If the solution of the particle is dominated by the gbest solution, assign the particle solution to gbest.
2. If the solution of the particle equals any solution in the non-dominated solution set, replace the non-dominated solution with the particle solution.
3. If the solution of the particle is dominated by the worst non-dominated solution and not equal to any non-dominated solution, set the worst non-dominated solution equal to the particle solution.

5 Computational results

The proposed PSO algorithm was verified by benchmark problems obtained from the OR-Library that were contributed by Carlier [2], Heller [8], and Reeves [23]. The test program was coded in Visual C++ and run 20 times on each problem using an Intel Pentium 4 3.0-GHz processor with 1 GB of RAM running Windows XP. We used four swarm sizes N (10, 20, 60, and 80) to test the algorithm during a pilot experiment. A value of N=80 was best, so it was used in all subsequent tests. The algorithm parameters were set as follows: c_1 and c_2 were tested over the range 0.1–0.7 in increments of 0.2, and the inertial weight w was reduced from $w_{\max}$ to $w_{\min}$ during the iterations. Parameter $w_{\max}$ was set to 0.5, 0.7, and 0.9 corresponding to $w_{\min}$ values of 0.1, 0.3, and 0.5. Settings of c_1=0.7, c_2=0.1, $w_{\max}$=0.7, and $w_{\min}$=0.3 worked best.

The proposed PSO algorithm was compared with five heuristic algorithms: CDS[1], NEH[17], RAJ[20], GAN-RAJ[6] and Laha[13]. We also coded these methods in Visual C++. The CDS heuristic [1] takes its name from its three authors and is a heuristic generalization of Johnson's algorithm. The process generates a set of $m-1$ artificial two-machine problems, each of which is then solved by Johnson's rule. In this study, we modified the original CDS and compared the makespan, mean flow time, and machine idle time of all $m-1$ generated problems. The non-dominated solution was selected to compare with the solutions obtained from our PSO algorithm. The other comparison was based on solutions determined by the NEH algorithm introduced by Nawaz et al. [17]. The NEH investigates $n(n+1)/2$ permutations to find near-optimal solutions. As we did for CDS, we modified the original NEH and compared the three objectives of all $n(n+1)/2$ sequences. We compared the non-dominated solution from these sequences with the solutions from our PSO.

The following two performance measures are used in this study: average-relative percentage deviation (ARPD) and maximum percentage deviation (MPD) where MS stands for makespan, TFT represents total flow time, MIT stands for machine idle time, H is the heuristic.

$$\mathrm{ARPD_{MS}} = \frac{100}{10}\sum_{i=1}^{10}\left(\frac{MS_{H,i}-\mathrm{BestMS}_i}{\mathrm{BestMS}_i}\right) \tag{3}$$

$$\mathrm{MPD_{MS}} = \mathrm{MAX}_{i=1..10}\left(\frac{MS_{H,i}-\mathrm{BestMS}_i}{\mathrm{BestMS}_i}\right)\times 100 \tag{4}$$

$$\mathrm{ARPD_{TFT}} = \frac{100}{10}\sum_{i=1}^{10}\left(\frac{\mathrm{TFT}_{H,i}-\mathrm{BestTFT}_i}{\mathrm{BestTFT}_i}\right) \tag{5}$$

$$\mathrm{MPD_{TFT}} = \mathrm{MAX}_{i=1..10}\left(\frac{\mathrm{TFT}_{H,i}-\mathrm{BestTFT}_i}{\mathrm{BestTFT}_i}\right)\times 100 \tag{6}$$

$$\mathrm{ARPD_{MIT}} = \frac{100}{10}\sum_{i=1}^{10}\left(\frac{\mathrm{MIT}_{H,i}-\mathrm{BestMIT}_i}{\mathrm{BestMIT}_i}\right) \tag{7}$$

$$\mathrm{MPD_{MIT}} = \mathrm{MAX}_{i=1..10}\left(\frac{\mathrm{MIT}_{H,i}-\mathrm{BestMIT}_i}{\mathrm{BestMIT}_i}\right)\times 100 \tag{8}$$

We tested our PSO on nine different problem sizes (n= 20, 50, 100 and m=5, 10, 20) from Taillard's [28] benchmarks. Table 1 compares the six methods using the

Table 1 Comparison of makespan(MS) for different heuristics

Problem size		NEH [17]		CDS [1]		RAJ [20]		GAN-RAJ [6]		Laha [13]		PSO	
n	m	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD
20	5	1.84	0.25	0.76	0.15	0.44	0.12	0.63	0.14	1.55	0.2	0.00	0.00
	10	1.78	0.23	0.71	0.12	0.85	0.17	0.83	0.14	1.50	0.20	0.00	0.00
	20	1.27	0.17	0.44	0.06	0.88	0.14	0.82	0.12	1.06	0.15	0.00	0.00
50	5	1.24	0.17	0.83	0.14	0.26	0.05	0.37	0.08	1.29	0.22	0.02	0.02
	10	1.28	0.19	0.59	0.08	0.48	0.09	0.53	0.10	1.29	0.18	0.01	0.01
	20	1.08	0.17	0.07	0.02	0.35	0.07	0.39	0.07	1.02	0.16	0.06	0.03
100	5	1.04	0.19	0.46	0.12	0.36	0.07	0.23	0.07	1.05	0.16	0.07	0.07
	10	0.28	0.06	0.47	0.07	0.29	0.06	0.24	0.04	0.89	0.13	0.01	0.01
	20	0.65	0.11	0.16	0.04	0.21	0.05	0.18	0.04	0.72	0.10	0.01	0.01

NEH Nawaz et al. [17], *CDS* Campbell et al. [1], *RAJ* Rajendran C [20], *GAN-RAJ* Gangadharan and Rajendran [6], *Laha* Laha and Chakraborty [12, 13], *PSO* proposed PSO)

Table 2 Comparison of total flow time (TFT) for different heuristics

Problem size		NEH [17]		CDS [1]		RAJ [20]		GAN-RAJ [6]		Laha [13]		PSO	
n	m	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD
20	5	0.65	0.17	1.71	0.27	1.70	0.31	1.88	0.34	4.43	0.61	1.28	0.20
	10	0.70	0.10	1.43	0.18	1.29	0.19	1.47	0.23	3.43	0.51	0.95	0.12
	20	0.59	0.14	1.23	0.18	1.27	0.21	1.31	0.24	2.29	0.30	0.82	0.12
50	5	0.11	0.07	2.48	0.56	2.56	0.51	2.58	0.53	5.86	0.94	2.48	0.44
	10	7.87	7.53	11.33	9.62	10.91	9.24	11.27	9.50	14.49	10.87	10.78	9.19
	20	0.39	0.09	1.55	0.20	1.58	0.20	1.60	0.19	3.18	0.40	1.44	0.17
100	5	0.27	0.27	2.24	2.24	3.59	3.59	3.00	3.00	5.56	5.56	2.60	2.60
	10	0.87	0.87	1.86	1.86	1.91	1.91	1.80	1.80	4.02	4.02	1.93	1.93
	20	1.39	1.39	1.65	1.65	1.73	1.73	1.65	1.65	2.83	2.83	1.59	1.59

(*NEH* Nawaz et al. [17], *CDS* Campbell et al. [1], *RAJ* Rajendran C [20], *GAN-RAJ* Gangadharan and Rajendran [6], *Laha* Laha and Chakraborty [12, 13], *PSO* proposed PSO)

Table 3 Comparison of machine idle time (MIT) for different heuristics

Problem size		NEH [17]		CDS [1]		RAJ [20]		GAN-RAJ [6]		Laha [13]		PSO	
n	m	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD
20	5	4.54	2.94	43.56	20.33	3.20	1.03	5.04	1.38	10.79	4.70	1.50	0.43
	10	3.87	0.83	15.03	1.94	8.07	1.48	7.93	1.42	9.92	1.76	0.00	0.00
	20	11.37	1.55	19.19	2.40	14.88	2.01	14.46	1.85	15.29	2.10	0.00	0.00
50	5	67.77	26.95	208.65	108.95	17.11	11.76	17.08	11.76	52.70	23.48	2.95	2.82
	10	1.92	0.56	10.59	1.74	4.74	0.68	4.91	0.70	6.92	1.24	0.26	0.18
	20	2.26	0.36	8.02	0.97	5.75	0.83	5.80	0.87	7.47	0.96	0.00	0.00
100	5	18.18	4.94	40.24	7.65	4.41	1.40	2.00	0.76	15.47	3.34	3.51	1.69
	10	1.96	0.43	9.54	1.38	1.92	0.38	1.65	0.41	5.47	0.98	0.15	0.09
	20	1.03	0.26	4.26	0.52	2.79	0.40	2.64	0.35	3.77	0.45	0.00	0.00

(*NEH* Nawaz et al. [17], *CDS* Campbell et al. [1], *RAJ* Rajendran C [20], *GAN-RAJ* Gangadharan and Rajendran [6], *Laha* Laha and Chakraborty [12, 13], *PSO* proposed PSO)

Table 4 Summation of MS, TFT and MIT for different heuristics

Problem size		NEH [17]		CDS [1]		RAJ [20]		GAN-RAJ [6]		Laha [13]		PSO	
n	m	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD
20	5	7.04	3.35	46.03	20.75	5.34	1.46	7.56	1.86	16.77	5.52	2.78	0.63
	10	6.36	1.16	17.18	2.25	10.21	1.83	10.23	1.79	14.85	2.46	0.95	0.12
	20	13.23	1.86	20.86	2.64	17.03	2.36	16.60	2.22	18.63	2.54	0.82	0.12
50	5	69.12	27.19	211.96	109.65	19.93	12.33	20.03	12.37	59.84	24.64	5.45	3.28
	10	11.08	8.28	22.51	11.44	16.13	10.00	16.71	10.30	22.70	12.29	11.04	9.38
	20	3.72	0.62	9.64	1.19	7.68	1.10	7.79	1.13	11.68	1.52	1.50	0.20
100	5	19.49	5.41	42.93	10.01	8.37	5.06	5.23	3.82	22.08	9.06	6.18	4.35
	10	3.11	1.36	11.87	3.32	4.12	2.35	3.69	2.25	10.38	5.13	2.08	2.02
	20	3.08	1.77	6.07	2.21	4.73	2.19	4.47	2.04	7.33	3.38	1.60	1.60

(*NEH* Nawaz et al. [17], *CDS* Campbell et al. [1], *RAJ* Rajendran C [20], *GAN-RAJ* Gangadharan and Rajendran [6], *Laha* Laha and Chakraborty [12, 13], *PSO* proposed PSO)

Table 5 Average CPU time (in seconds)

n	m	NEH	CDS	RAJ	GANRAJ	Laha [12]	PSO
20	5	0.0016	0.0031	0.0047	0.0014	0.0012	1.6641
	10	0.0015	0.0093	0.0094	0.0015	0.0015	2.0547
	20	0.0047	0.0109	0.0094	0.0031	0.0047	2.8078
50	5	0.0140	0.0016	0.0156	0.0047	0.0047	4.4906
	10	0.0234	0.0032	0.0297	0.0047	0.0063	5.3047
	20	0.0500	0.0078	0.0539	0.0078	0.0062	7.1593
100	5	0.0860	0.0016	0.0844	0.0047	0.0047	11.9094
	10	0.1750	0.0046	0.1750	0.0047	0.0078	13.4906
	20	0.3750	0.0078	0.3656	0.0079	0.0141	17.0079

(*NEH* Nawaz et al. [17], *CDS* Campbell et al. [1], *RAJ* Rajendran C [20], *GAN-RAJ* Gangadharan and Rajendran [6], *Laha* Laha and Chakraborty [12, 13], *PSO* proposed PSO)

Table 6 Comparison of total flow time (TFT) for different heuristics in ARPD

n	m	NEH	CDS	RAJ	GANRAJ	Laha	LR	SA	H-1	H-2	PSO
20	5	0.65	1.71	1.70	1.88	4.43	0.24	1.17	0.16	0.20	1.28
	10	0.70	1.43	1.29	1.47	3.43	0.09	0.72	0.01	0.01	0.95
	20	0.59	1.23	1.27	1.31	2.29	0.15	0.66	0.12	0.07	0.82
50	5	0.11	2.48	2.56	2.58	5.86	0.56	1.78	0.55	0.54	2.48
	10	7.87	11.33	10.91	11.27	14.49	8.06	1.24	7.97	7.89	10.78
	20	0.39	1.55	1.58	1.60	3.18	0.15	1.10	0.08	0.09	1.44
100	5	0.27	2.24	3.59	3.00	5.56	0.43	1.59	0.43	0.43	2.60
	10	0.87	1.86	1.91	1.80	4.02	0.04	1.24	0.03	0.03	1.93
	20	1.39	1.65	1.73	1.65	2.83	0.08	1.13	0.01	0.02	1.59

NEH Nawaz et al. [17], *CDS* Campbell et al [1], *RAJ* Rajendran C [20], *GAN-RAJ* Gangadharan R, Rajendran C [6], *Laha* Laha and Chakraborty [12, 13], *LR* Liu J, Reeves CR [16], *SA* Chakravarthy K, Rajendran C [3], *H-1 and H-2* Laha D, Chakraborty UK [12, 13], *PSO* proposed PSO)

Table 7 Comparison of total flow time (TFT) for different heuristics in MPD

n	m	NEH	CDS	RAJ	GANRAJ	Laha	LR	SA	H-1	H-2	PSO
20	5	0.17	0.27	0.31	0.34	0.61	0.12	0.21	0.11	0.12	0.20
	10	0.10	0.18	0.19	0.23	0.51	0.01	0.12	0.00	0.01	0.12
	20	0.14	0.18	0.21	0.24	0.30	0.05	0.12	0.05	0.05	0.12
50	5	0.07	0.56	0.51	0.53	0.94	0.25	0.38	0.25	0.25	0.44
	10	7.53	9.62	9.24	9.50	10.87	7.92	0.19	7.87	7.82	9.19
	20	0.09	0.20	0.20	0.19	0.40	0.04	0.16	0.04	0.04	0.17
100	5	0.27	2.24	3.59	3.00	5.56	0.43	1.59	0.43	0.43	2.60
	10	0.87	1.86	1.91	1.80	4.02	0.04	1.24	0.03	0.03	1.93
	20	1.39	1.65	1.73	1.65	2.83	0.08	1.13	0.01	0.02	1.59

NEH Nawaz et al. [17], *CDS* Campbell et al [1], *RAJ* Rajendran C [20], *GAN-RAJ* Gangadharan R, Rajendran C [6], *Laha* Laha and Chakraborty [12, 13], *LR* Liu J, Reeves CR [16], *SA* Chakravarthy K, Rajendran C [3], *H-1 and H-2* Laha D, Chakraborty UK [12, 13], *PSO* proposed PSO)

ARPD and MPD. Table 1 show that the proposed PSO outperforms for almost all problem instances in the makespan object. The comparison of TFT object is revealed in Table 2. It shows the ARPD and MPD of six heuristics and the Laha's algorithm performs better. We have given the comparison of MIT in Table 3 that indicates the proposed PSO can get better solution. At last, we aggregate the results of three objects in order to show the performance of the proposed PSO to solve the multi-objectives problems. We observed that the PSO performed better than other five heuristics. Table 4 shows the superior performance of the proposed PSO in terms of the three simultaneous objectives. The computation cost is demonstrated on Table 5. The proposed PSO spends more CPU time than other construct heuristic because of the proposed PSO is an evolutionary algorithm.

In addition, we compare TFT of benchmarks by more algorithms—Liu and Reeves[16] (LR), Chakravarthy-Rajendran [3], simulated annealing-bases approach (SA) and Laha and Chakraborty [12] (H-1 and H-2). The results are shown in Table 6 for ARPD and Table 7 for MPD. We can observe that the H-1 and H-2 perform better than other algorithms while only one object TFT is considered.

6 Conclusion

Many flowshop scheduling problem studies have been conducted in the past. However, the objective of most of these has been the minimization of the maximum completion time (i.e., the makespan). In the real world, there exist other objectives, such as minimization of machine idle time that might help improve efficiency and reduce production costs. PSO, which was inspired by the behavior of birds and fish, has certain advantages, including simple structure, easy implementation, immediate accessibility, short search time, and robustness. However, there has been limited study of PSO to address the multiple objectives found in the flowshop scheduling problem. We have therefore presented a PSO method for solving a flowshop scheduling problem with multiple objectives, including minimization of makespan, mean flow time, and machine idle time.

PSO was originally proposed for continuous optimization problems. We modified the representation of particle position, particle movement, and particle velocity to make PSO suitable for flowshop scheduling, which is a combinational problem. In addition, we used a mutation operator in our PSO algorithm. We also incorporated the concept of Pareto optimality to measure the performance of multiple objectives rather than using a weighted fitness function. Another necessary adjustment to the original PSO, required to maintain the Pareto optimal solution, was the external Pareto optimal set used to produce a limited size of non-dominated solutions. We also used a diversification strategy in our PSO algorithm. The results demonstrated that the proposed PSO could produce more optimal solutions than other heuristics (CDS, NEH, RAJ, GAN-RAJ, and Laha). The ARPD and MPD of each problem scenario in our PSO algorithm were less than those methods. The results of our performance measurement also revealed that the proposed PSO algorithm outperformed the heuristics in minimizing the makespan, mean flow time, and total machine idle time.

In future research, we will attempt to apply our PSO to other shop scheduling problems with multiple objectives. Possible topics for further study include modification of the particle position representation, particle movement, and particle velocity. Issues related to Pareto optimality, such as a solution maintenance strategy and performance measurement, are also topics worthy of future study.

References

1. Campbell HG, Dudek RA, Smith ML (1970) A heuristic algorithm for the n-job m-machine sequencing problem. Manage Sci 16:B630–B637. doi:10.1287/mnsc.16.10.B630
2. Carlier J (1978) Ordonnancements à contraintes disjonctives. RAIRO Rech Oper. Oper Res 12:333–351
3. Chakravarthy K, Rajendran C (1999) A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization. Prod Plann Contr 10:707–714. doi:10.1080/095372899232777
4. Eren T, Güner E (2007) The tricriteria flowshop scheduling problem. Int J Adv Manuf Technol 36:1210–1220. doi:10.1007/s00170-007-0931-1
5. Gupta JND, Stafford JEF (2006) Flowshop scheduling research after five decades. Eur J Oper Res 169:699–711. doi:10.1016/j.ejor.2005.02.001
6. Gangadharan R, Rajendran C (1993) Heuristic algorithms for scheduling in no-wait flow shop. Int J Prod Econ 32:285–290. doi:10.1016/0925-5273(93) 90042-J
7. Hejazi SR, Saghafian S (2005) Flowshop- scheduling problems with makespan criterion: a review. Int J Prod Res 43:2895–2929. doi:10.1080/0020754050056417
8. Heller J (1960) Some numerical experiments for an MxJ flow shop and its decision- theoretical aspects. Oper Res 8:178–184. doi:10.1287/opre.8.2.178
9. Jarboui B, Ibrahim S, Siarry P, Rebai A (2008) A combinational particle swarm optimisation for solving permutation flowshop problems. Comput Ind Eng 54:526–538. doi:10.1016/j.cie.2007.09.006
10. Kennedy J, Eberhart R (1995) Particle swarm optimization. Proc IEEE Int Conf Neural Netw 1995:1942–1948. doi:10.1109/ICNN.1995.488968
11. Knowles JD, Corne DW (1999) The Pareto archived evolution strategy: a new baseline algorithm for multi-objective optimization. In: Congress on Evolutionary Computation, Washington, DC, IEEE Service Center, 98–105
12. Laha D, Chakraborty UK (2008) An efficient heuristic approach to total flowtime minimization in permutation flowshop scheduling. Int J Adv Manuf Technol 38:1018–1025. doi:10.1007/s00170-007-1156-z
13. Laha D, Chakraborty UK (2009) A constructive heuristic for minimizing makespan in no-wait flow shop scheduling. Int J Adv Manuf Technol 41:97–109. doi:10.1007/s00170-008-1545-0

14. Lian Z, Gu X, Jiao B (2008) A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan. Chaos Solitons Fractals 35:851–861. doi:10.1016/j.chaos.2006.05.082
15. Liu B, Wang L, Jin YH (2007) An effective PSO-based memetic algorithm for flow shop scheduling. IEEE Trans Syst Man Cybern C 37:18–27. doi:10.1109/TSMCB.2006.883272
16. Liu J, Reeves CR (2001) Constructive and composite heuristic solutions to the P//∑Ci scheduling problem. Eur J Oper Res 132:439–452. doi:10.1016/S0377-2217(00) 00137-5
17. Nawaz M, Enscore JR, Ham I (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11:91–95. doi:10.1016/0305-0483(83) 90088-9
18. Pasupathy T, Rajendran C, Suresh RK (2006) A multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs. Int J Adv Manuf Technol 27:804–815. doi:10.1007/s00170-004-2249-6
19. Ponnambalam SG, Jagannathan H, Kataria M (2004) A TSP-GA multi-objective algorithm for flow-shop scheduling. Int J Adv Manuf Technol 23:909–915. doi:10.1007/s00170-003-1731-x
20. Rajendran C (1994) A no-wait flow shop scheduling heuristic to minimize makespan. J Oper Res Soc 45:472–478
21. Rajendran C, Ziegler H (2004) Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. Eur J Oper Res 155:426–438. doi:10.1016/S0377-2217(02) 00908-6
22. Rahimi-Vahed A, Mirghorbani S (2007) A multi-objective particle swarm for a flow shop scheduling problem. J Comb Optim 13:79–102. doi:10.1007/s10878-006-9015-7
23. Reeves CR (1995) A genetic algorithm for flowshop sequencing. Comput Oper Res 22:5–13. doi:10.1016/0305-0548(93) E0014-K
24. Ruiz R, Maroto C (2004) A comprehensive review and evaluation of permutation flowshop heuristics. Eur J Oper Res 165:479–494. doi:10.1016/j.ejor.2004.04.017
25. Sha DY, Hsu CY (2006) A hybrid particle swarm optimization for job shop scheduling problem. Comput Ind Eng 51:791–808. doi:10.1016/j.cie.2006.09.002
26. Sha DY, Hsu CY (2008) A new particle swarm optimization for the open shop scheduling problem. Comput Oper Res 35:3243–3261. doi:10.1016/j.cor.2007.02.019
27. Stützle T (1998) Applying iterated local search to the permutation flow shop problem. Tech Rep, AIDA-98-04, FG Intellektik, TU Darmstadt.
28. Taillard E (1993) Benchmarks for basic scheduling problems. Eur I Oper Res 64:278–285. doi:10.1016/0377-2217(93) 90182-M
29. Tasgetiren MF, Liang YC, Sevkli M, Gencyilmaz G (2007) A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. Eur J Oper Res 177:1930–1947. doi:10.1016/j.ejor.2005.12.024
30. Yagmahan B, Yenisey MM (2008) Ant colony optimization for multi-objective flow shop scheduling problem. Comput Ind Eng 54:411–420. doi:10.1016/j.cie.2007.08.003
31. Zhang H, Li X, Li H, Huang F (2005) Particle swarm optimization-based schemes for resource-constrained project scheduling. Auto Const 14:393–404. doi:10.1016/j.autcon.2004.08.006
32. Zizter E, Laumanns M, Thiele L (2001) SPEA2: Improving the strength Pareto evolutionary algorithm. Computer Engineering and Networks Laboratory (TIK) – Report 103 Sept 2001.