*Journal of Optimization in Industrial Engineering*
*Vol.12, Issue 2, Summer & Autumn 2019, 107- 119*
*DOI: 10.22094/JOIE.2018.671.1433*

Research Article

# Diversified Particle Swarm Optimization for Hybrid Flowshop Scheduling

**Javad Behnamian***

*Department of Industrial Engineering, Faculty of Engineering, Bu-Ali Sina University, Hamedan, Iran*
Received 23 October 2016; Revised 20 April 2017; Accepted 14 February 2018

**Abstract**

The aim of this paper is to propose a new particle swarm optimization algorithm to solve a hybrid flowshop scheduling with sequence-dependent setup times problem, which is of great importance in the industrial context. This algorithm is called diversified particle swarm optimization algorithm which is a generalization of particle swarm optimization algorithm and inspired by an anarchic society whose members behave anarchically to improve their situations. Such anarchy lets the algorithm explore the solution space perfectly and prevent falling in the local optimum traps. Besides, for the first time, for the hybrid flowshop, we proposed eight different local search algorithms and incorporate them into the algorithm in order to improve it with the help of systematic changes of the neighborhood structure within a search for minimizing the makespan. The proposed algorithm was tested and the numerical results showe that the proposed algorithm significantly outperforms other effective heuristics recently developed.

***Keywords**:* Particle swarm optimization; Scheduling; Sequence-dependent; Hybrid flowshop.

## 1. Introduction

Particle swarm optimization (PSO) algorithm has been successfully applied to a variety of problems, such as continuous and combinatorial optimization problems, artificial neural network training, and multi-objective optimization problems (García-Villoria and Pastor 2009). However, there are several difficulties to apply PSO to discrete optimization problems because Kennedy and Eberhart (1995) originally developed it for continuous problems. Another shortcoming of PSO is its premature convergence. To overcome these shortcomings, in this paper, a diversified particle swarm optimization (DPSO) algorithm as a powerful generalization of PSO algorithm is introduced. DPSO algorithm is originally inspired by the concept of a society whose members behave anarchically to improve their situations. In DPSO algorithm, the particles are fickle, and their fickleness increases when their situations become worse. They also occasionally behave irregularly and move towards worse particles of swarm or worse positions which have been previously visited. DPSO algorithm of such anarchic particles explores the solution space perfectly and avoids being trapped into the local optima.

A hybrid flowshop can model the process industry, including chemical, pharmaceutical, oil, food, tobacco, textile, paper, and metallurgical industry. In fact, the hybrid flowshop is a flowshop which consists of parallel machines in production stages. The flow of products is unidirectional and each product is processed at only one facility in each stage and at one or more stages. In this production environment, we also assume that there is a setup time for jobs which is sequence-dependent. Scheduling problems with sequence-dependent setup times are among the most difficult classes of scheduling problems (Pinedo 2012) and sequence-dependent setup scheduling of a hybrid flowshop system is even more challenging (Allahverdi 2015). In this paper, following Ahmadi-Javid (2011), we present an DPSO algorithm for the sequence-dependent hybrid flowshop scheduling problem (SDST-HFSP).

The paper is organized as follows. Section 2 gives the literature review of the related papers. Section 3 introduces the problem description. Section 4 presents the DPSO algorithm implemented for SDST-HFSP. Numerical study is given in Section 5. Finally, Section 6 is devoted to conclusions and future studies.

## 2. Literature Review

In this section, first, a brief review of continuous and discrete PSO algorithms is given in sub-sections 2.1 and 2.2. Then, in sub-section 2.3, some diversification approaches for PSO algorithm are reviewed. Finally, in sub-section 2.4, the (meta)heuristics of the hybrid flowshops are reviewed.

### *2.1. Continuous PSO algorithm*

There are several papers which have reported successful application of PSO algorithm for optimization problems. Some successful applications of PSO algorithm are as follows: inventory planning (Tsou, 2008), robotics (Coelho & Sierakowski, 2008), water supply systems (Montalvo et al., 2008), image segmentation (Maitra & Chatterjee, 2008), classification (Marinakis et al., 2008),

*Corresponding author Email address: behnamian@basu.ac.ir

support vector machines (Lin et al., 2008), resource allocation (Yang et al., 2008), communication networks (Huang et al., 2008), dynamic question generation (Cheng et al., 2009), reliability-redundancy optimization (Coelho, 2009), clustering (Chen & Zhao, 2009), mining breast cancer pattern (Yeh et al., 2009), gate array placement (El-Abd et al., 2010), quality control and inspection (Sun, 2009), identify Wiener model (Tang et al., 2010), assembly sequence planning (Wang and Liu, 2010), and others.

*2.2. Discrete PSO algorithm*

Kennedy and Eberhart (1997) proposed the first discrete PSO algorithm which is characterized by a binary solution representation and a stochastic velocity model. Several binary solution representation models were proposed and tested in Mohan and Al-kazemi (2001). Another approach for tackling discrete optimization problems by PSO algorithm was also proposed by Laskari et al. (2002) which is based on the truncation of the real values to their nearest integer. Some other papers related to the discrete PSO algorithm are as follows: n-queens problem (Hu et al. 2003), polygonal approximation of digital curves (Yin, 2004), no-wait flowshop (Liu et al., 2005), vehicle routing problem (Chen et al., 2006), resource-constrained project scheduling (Zhang et al., 2006), job shop (Sha and Hsu, 2006), transmission network expansion planning (Xiong et al., 2007), data clustering (Karthi et al., 2008), estimation of polygonal approximation's distribution (Wang et al., 2009), traveling salesman's problem (Król and Drożdżowski, 2010), and flowshop (Ahmadi-Javid, 2011).

*2.3. Diversity mechanisms for PSO algorithm*

There are different ways to prevent premature convergence and to make diversity in PSO algorithm. Loovbjerg (2002) used self-organized PSO algorithm. In this algorithm, when two particles are too close to one another, a variable called the "critical value" is incremented. When it reaches the criticality threshold, the particle disperses its criticality to other particles that are close to it and relocates itself. Hu and Eberhart (2002) introduced the use of a dynamic neighborhood to the PSO algorithm. In this algorithm, having updated swarm, the particles update their neighborhoods. Xie et al. (2002) added negative entropy to the particle swarm in order to discourage premature convergence (excessively rapid convergence towards a poor quality local optimum). In some conditions, they weighted the velocity and, in some conditions, the particle's locations are set by some random values. Blackwell and Bentley (2002) reduced the attraction of the swarm center to prevent the particles clustering too tightly in one region of the search space. For multi-objective optimization problem, Parsopoulos and Vrahatis (2004) suggested theuse of multiple swarms where the number of swarms is equal to that of objective functions. Each swarm searches one objective function and the best solution found by a swarm is fed to another swarm to direct the search of the particles of that swarm. Clerc (2006) dynamically changed the size of the swarm according to the performance of the algorithm. The size of the swarm is important because too few particles will cause the algorithm to converge prematurely to a local optimum, while too many particles will slow down the algorithm. Gaafar et al. (2008) combined PSO algorithm with a genetic algorithm to schedule in an agile environment. Xiang et al. (2008) proposed a new method in which particles have time-delay to control the process of information diffusion. Sadati et al. (2009) used knowledge-based cooperative strategy to particle diversification. Jie et al. (2008) hybridized PSO algorithm with a simulated annealing algorithm. García-Villoria and Pastor (2009) added a random velocity according to the incongruity of the population in the PSO algorithm. Koua et al. (2009) enhanced particles' society by adding a co-evolutionary strategy and evolving direction. In the co-evolutionary strategy, a deterministic selection strategy is used to ensure the diversity of the population. In this study, the infeasible solution was properly accepted as a feasible solution and the proposed diversity mechanism was helpful to guide the search direction of infeasible solution towards the feasible region. Chen (2011) introduced two layers of PSO algorithm with $M$ swarms of particles and one swarm of particles in the bottom and top layers, respectively.

*2.4. Hybrid flowshops*

Recently, several heuristics have been developed for hybrid flowshops. Botta-Genoulaz (2000) proposed several heuristics for a flowshop with multiple identical machines per stage, positive time lags, and realistic constraints as well as sequence-independent setup and removal times. Azizoglu et al. (2001) considered the total flow time measure in a multi-stage hybrid flowshop and suggested a branch and bound algorithm that gives the optimal solutions for moderate-size problems. Harjunkoski and Grossmann (2002) considered setup times that only depend on the machine and not on the job. Lee et al. (2003) developed a dispatching rule-based approach for the HFSP in a printed circuit board manufacturing system. Kurz and Askin (2003) compared several methods for a makespan minimization problem with sequence-dependent setup times in which jobs are allowed to skip stages. They also developed an integer model, some heuristics, and a random keys genetic algorithm (RKGA) for SDST flexible flowshop (Kurz and Askin 2004). Wardono and Fathi (2004) developed a tabu search algorithm for a multi-stage parallel machine problem with limited buffers. Another research which addressed the real-world industries' problems was studied by Andres et al. (2005). They considered the problem of products grouping in a tile industry. They proposed some heuristic and metaheuristic algorithms for a three-stage HFSP with sequence-dependent setup times. For HFSP, Tang et al. (2006) proposed a new Lagrangian relaxation algorithm based on stage decomposition for minimizing the total weighted completion time. Janiak et al. (2007)

proposed some approximation algorithms for the HFSP with cost-related criterion. In this paper, the scheduling criterion consists of three parts: total weighted earliness, total weighted tardiness, and total weighted waiting time. An improved ant colony optimization for hybrid flowshop scheduling was proposed by Alaykiran et al. (2007) to minimize $C_{\max}$ criterion. In order to achieve better results, they conducted a parameter optimization study. Kim et al. (2008) focused on the scheduling problem of minimizing makespan for a given set of jobs in a two-stage hybrid flowshop subject to a product-mix ratio constraint. Ying (2008) proposed an iterated greedy heuristic to minimize makespan in a multistage hybrid flowshop with multiprocessor tasks. In this research, to validate and verificate the proposed heuristic, computational experiments were performed on two benchmark problem sets. Jina et al. (2006) considered the multistage hybrid flowshop scheduling problem. To minimize the makespan, based on simulated annealing and the variable-depth search, they proposed an optimization procedure. This study reveals that the proposed metaheuristic in comparison with Johnson (Johnson et al. 1989) and shortest processing time (SPT) rules and tabu search was an efficient algorithm. Tseng and Liao (2008) proposed a PSO algorithm for hybrid flowshop scheduling. In this study, computational results show that the PSO algorithm outperformed the genetic algorithms and an ant colony system algorithm. Wang and Tang (2009) hybridized tabu search algorithm with scatter search algorithm to solve the hybrid flowshop scheduling with finite intermediate buffers, whose objective is to minimize the sum of weighted completion time. They showed that this hybrid heuristic can provide good solutions compared to lower bounds, NEH (Nawaz et al. 1983) and genetic algorithm (GA). Behnamian et al. (2009) considered the problem of sequence-dependent setup time hybrid flowshop scheduling with the objectives of minimizing the makespan and sum of the earliness and tardiness of jobs, and presented a three-phase multi objective method.

Naderi et al. (2011) investigated the scheduling flexible flowshops subject to periodic preventive maintenance on machines to minimize the makespan. They proposed a genetic algorithm, an artificial immune system, and some constructive heuristics to tackle the problem. Behnamian et al. (2012) proposed a hybrid metaheuristic algorithm including ant system, simulated annealing, and variable neighborhood search to determine a scheduling for hybrid flowshop problem that minimizes the makespan of jobs. Recently, Li et al. (2014) proposed a hybrid variable neighborhood search algorithm that combines chemical-reaction optimization and estimation of distribution to minimize the maximum completion in the hybrid flowshop scheduling problems.

This paper proposed a new algorithm to solve the hybrid flowshop scheduling with sequence-dependent setup times problem. The proposed algorithm is a generalization of particle swarm optimization and inspired by the concept of an anarchic society whose members behave anarchically to improve their situations. Such anarchy lets algorithm explore the solution space perfectly and prevents falling in local optimum traps. Besides, we proposed eight different local search algorithms and incorporated them into the algorithm in order to improve it by systematic changes of the neighborhood structure within a search.

## 3. Problem Description

Since different constraints and assumptions can result in different scheduling problems in hybrid flowshops, we clearly mention our assumptions in the following:

1. All parameters are known deterministically when the scheduling is undertaken.
2. Stage $t$ has a set of $m^t$ identical machines in capability and processing rate.
3. Each stage has at least one machine, and at least one stage must have more than one machine.
4. A job which has once started on the machine must be completed without interruption.
5. A job can wait between two stages and the intermediate storages are unlimited.
6. A job can be processed by each one of the machines in each stage, and it will be processed by a single machine in each stage.
7. A machine can process only one job at a time.
8. There is no travelling time between two stages.
9. The machines are available at all times if they are not busy. Also, there is no breakdowns or scheduled/ unscheduled maintenance.
10. Having been released from the previous stage, each job is available for processing at a stage immediately.

## 4. DPSO Algorithm Framework

Diversified particle swarm optimization algorithm is a generalization of PSO algorithm. DPSO algorithm is inspired by an anarchic society whose members behave anarchically to improve their situations. In DPSO algorithm, the particles are fickle, and their fickleness increases as their situations become worse. They also occasionally behave irregularly and move towards the worse particles or worse positions which they have been previously visited. DPSO algorithm by such anarchic particles is able to search the solution space perfectly and prevent falling in the local optimum traps. In the following, we will present the DPSO algorithm framework. Note that this framework is general and presented for any optimization problem.

The basic notation of DPSO algorithm is quite similar to standard PSO algorithm. Let $S$ be a solution space and $f : S \to \Re$ be a function which should be minimized on $S$. Consider a swarm with *N* particles for solution space searching. For particle $i$ in iteration $k$, two elements in $S$ are defined as follows:

$X_i(k)$: Position,

$P_i(k)$: The best personal previously visited position, called P-best.

Also, all particles are aware of the best global position visited by all particles in iteration $k$ denoted by $G_i(k)$ and called G-best.

*4.1. The proposed algorithm*

To show that the DPSO algorithm framework can be implemented for discrete problems, in this section, we present a DPSO algorithm for the hybrid flowshop scheduling problem with sequence-dependent setup time and the objective of minimizing the makespan. Since the SDST single machine scheduling problem is equivalent to a traveling-salesman problem (Pinedo 2012) and is NP-hard, SDST-HFSP is also NP-hard. SDST-HFSP is broadly used for modeling the process industries such as chemical, pharmaceutical, oil, food, tobacco, textile, paper, and metallurgical industry. For a comprehensive survey, we refer to the recent review paper done by Allahverdi (2015).
The most important issue in applying DPSO algorithm to schedule problems is to find a suitable method to relate job sequences and positions of the particles. A suitable method is to represent each scheduling solution by a chromosome. This method enables us to use genetic operators, such as crossover and mutation, to simply generate movement policies. In the following subsections, first, we will explain our solution representation, and then present settings of the DPSO algorithm.

*4.2. Solution representation*

The proposed representation of DPSO algorithm to solve hybrid flowshop scheduling is based on coding all jobs as blocks in a $n+(m^1-1)$ string in which *n* and $m^1$ are the number of jobs and number of machines at the first stage of production line, respectively. In this type of representation, the sequence of jobs is represented by the numbers of blocks from the left to right and $(m^1-1)$ asterisks "*" are used to differentiate one machine from another.
An example of the representation is shown in Figure 1.

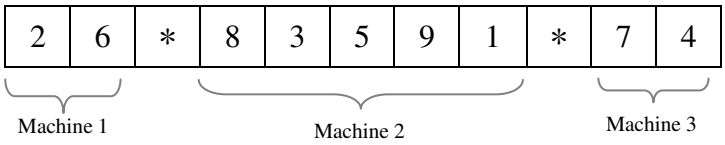


Fig. 1. Solution representation

In this example, there are nine jobs with three machines at the first stage. As shown in this figure, jobs 2 and 6 with order 2→6 are assigned to machine 1; jobs 7 and 4 are assigned to machine 3, and other jobs are assigned to machine 2.

*4.3. Swarm initialization*

Any available method for generating a feasible solution would be sufficient for our algorithm. Random initializing for constructive metaheuristics is a common and popular procedure (Talbi 2009). To generate random positions for the particles of the initial swarm, the procedure is as follows:
Generate $(m^1-1)$ asterisks "*" and randomly assign them to the blocks, a chromosome with length of $n+(m^1-1)$ blocks, so that at least one unfilled block must bebetween the asterisks . Then, it assigns numbers 1 to $n$ to the rest of the unfilled blocks. Note that the size of the swarm depends on the solution space.

*4.4. Movement in the next iteration*

Each particle has a planning procedure to decide how to move and change its position in the next iteration. To this end, each particle provides three movement policies and then combines them to determine its position in the next iteration. These movement policies are generated as follows:

*4.4.1. Movement policy based on current position:* $\mathrm{MP}_i^{\mathrm{Current}}(k)$

The first movement policy in iteration $k$ is denoted by $\mathrm{MP}_i^{\mathrm{Current}}(k)$ and is chosen based on the current position. Each particle is fickle and its position is determined based on its current position. In general, the movement policy $\mathrm{MP}_i^{\mathrm{Current}}(k)$ is a neighboring method in which the particles can choose a different neighborhood method. For this reason, the *G-best* can be appropriate in order to prevent falling in the local optimum trap. In this regard, *fickleness index* $FI_i(k)$ for particle $i$ in iteration $k$ is defined as Equation (1). The fickleness of each particle may depend on its relative situation to G-best or its P-best. Therefore, for each particle, the fickleness rate is defined as

$$FI_i(k)=1-\frac{f(G_i(k))}{f(X_i(k))} \quad (1)$$

where objective function $f$ is the makespan of decoded solution and *G* and *X* are G-best and a position of particles, respectively.

Now, we consider the two following cases:

**Case A: Particle** $i \neq$ G-best

Based on $FI_i(k)$, particle $i \neq$ G-best generates its movement policy $\mathrm{MP}_i^{\mathrm{Current}}(k)$ as follows:

$$\mathrm{MP}_i^{\mathrm{Current}}(k) = \begin{cases} \text{Apply LS1} & \text{for} \quad 0 \le FI_i(k) < 0.5 \\ \text{Apply LS2} & \text{for} \quad 0.5 \le FI_i(k) \le 1 \end{cases}$$

.

**Case B: Particle** $i =$ G-best

Particle $i =$ G-best selects randomly one of the following movement policies:

$$\mathrm{CMP}_{i=\mathrm{G-best}}^{\mathrm{Current}}(k) = \begin{cases} \text{Apply } LS\,3 \\ \text{Apply } LS\,4 \\ \text{Apply } LS\,5 \end{cases}$$

*4.4.2. Movement policy based on other particles:* $\mathrm{MP}_i^{\mathrm{Swarm}}(k)$

The second movement policy in iteration $k$ is denoted by $\mathrm{MP}_i^{\mathrm{Swarm}}(k)$ and is chosen based on the positions of other particles. It is logical and regular that each particle generates its movement policy $\mathrm{MP}_i^{\mathrm{Swarm}}(k)$ based on G-best, but since particles are anarchic, it may select other particles (or a number of them) to generate a movement policy. Hence, for all particles, we define external irregularity rates $EI_i(k)$ as

$$EI_i(k) = 1 - e^{-|CV(k)|} \qquad (2)$$

in which $CV(k)$ is the coefficient of variation of all particles fitness, *i.e.*, $f(X_1(k)), \ldots, f(X_N(k))$. This means that if the diversity of the swarm increases, the particles like to behave more irregularly, which is an expected behavior for anarchic particles. We use the following scenario for this policy:

$$\mathrm{MP}_i^{\mathrm{Swarm}}(k) = \begin{cases} \text{Apply } LS\,6 & 0 \le EI_i(k) < 0.5 \\ \text{Apply } LS\,7 & 0.5 \le EI_i(k) < 1 \end{cases}$$

*4.4.3. Movement policy based on past positions:* $\mathrm{MP}_i^{\mathrm{Past}}(k)$

The third movement policy in iteration $k$ is denoted by $\mathrm{MP}_i^{\mathrm{Past}}(k)$ and is chosen based on the past positions visited personally. It is more normal that each particle generates movement policy $\mathrm{MP}_i^{\mathrm{Past}}(k)$ based on P-best, but because in our proposed algorithm, particles are lawless, it may select one of the past positions (or a number of them) to generate a movement policy. Hence, we define *internal irregularity rate* $II_i(k)$ for particle $i$ in iteration $k$, and we assume that with probability $EI_i(k)$, particle $i$ behaves irregularly. It is assumed that, for all particles, the internal irregularity rates $II_i(k)$ are zeros, which means that particles generate policies $\mathrm{MP}_i^{\mathrm{Past}}(k)$ only based on their P-bests. Movement policy $\mathrm{MP}_i^{\mathrm{Past}}(k)$ is defined as:

$$\mathrm{MP}_i^{\mathrm{Past}}(k) = \text{Apply } LS\,8$$

*4.4.4. Combination rule*

After applying three components of velocity (inertia, P-best, G-best), there are three solutions which must be combined to update the particles position. In this paper, three mechanisms are proposed and the better one must be selected in fine tuning. These mechanisms are as follows:

- Serial: serial implementation $\{S_i \rightarrow S_p \rightarrow S_s\}$,
- Elitism: minimum$\{S_i, S_p, S_s\}$, and
- Crossover: crossover among $\{S_i, S_p, S_s\}$.

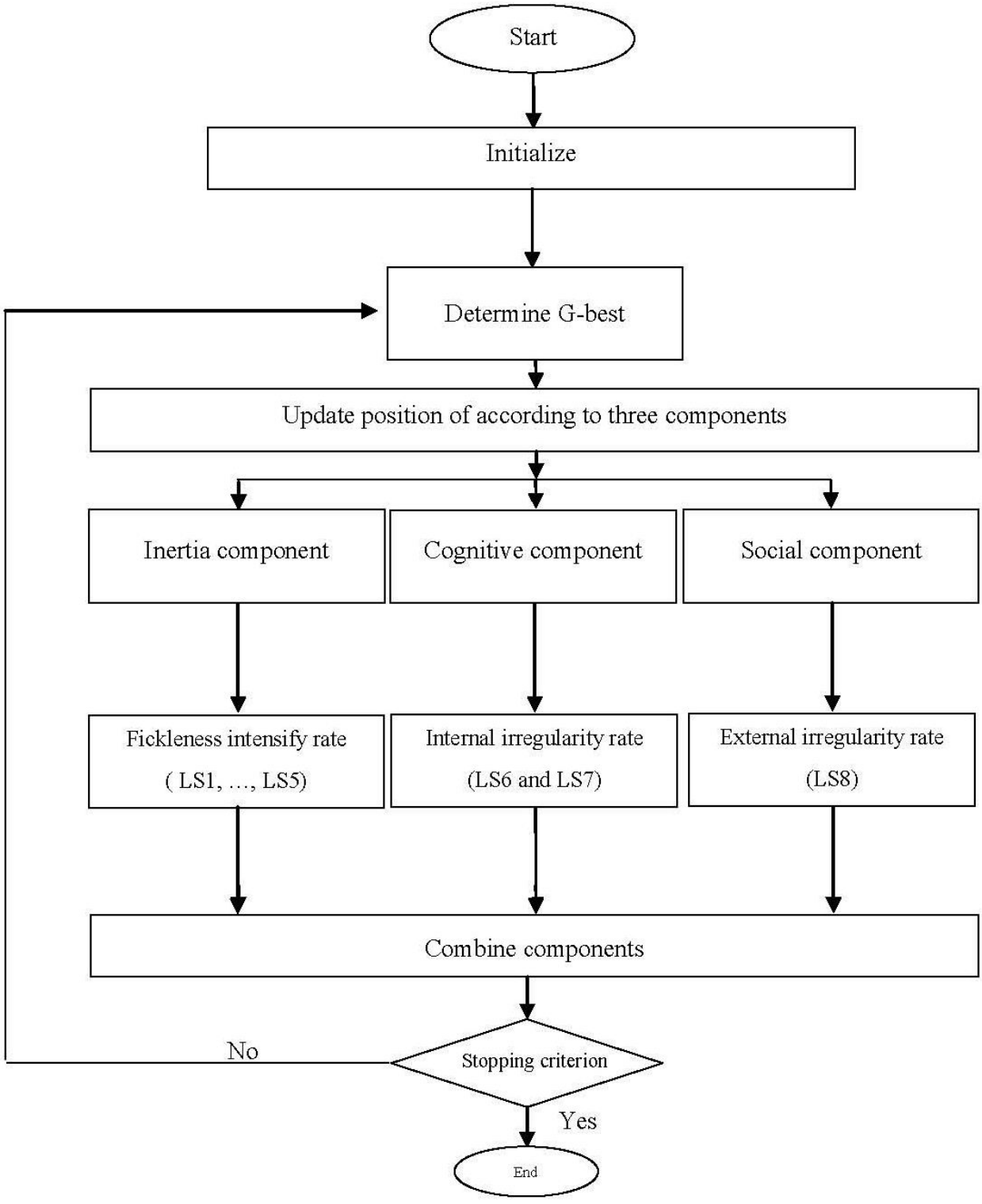


Fig. 2. Flowchart of diversified particle swarm optimization (DPSO) with eight local searches

*4.5. Local searches*

In this paper, we proposed several neighborhood search structures. Eight-type procedures are introduced in the following manner as local searches:

**Algorithm 1.** LS1:

1: Choose a machine $m$ randomly;
2: Choose two jobs $j_1$ and $j_2$ randomly from machine $m$;
3: Swap jobs $j_1$ and $j_2$.

**Algorithm 2.** LS2:

- Choose a machine $m$ randomly;
- Choose job $j$ and a valid position '*pos*' from machine $m$ randomly;
- Transfer job $j$ at the position *pos*.

**Algorithm 3. LS3:**

1: Choose machine $m$ randomly;
2: Choose cutting point from machine $i$ randomly to divide the sequence of jobs into two parts;
3: Swap two parts.

**Algorithm 4. LS4:**

1: Choose two machines $m_1$ and $m_2$ randomly;
2: Choose job $j_1$ in $m_1$ and job $j_2$ in $m_2$ randomly;
3: Swap jobs $j_1$ and $j_2$.

**Algorithm 5.** LS5:

1: Choose one job $j_1$ and one machine $m_2$ randomly, where $j_1$ does not belong to $m_2$;
2: Choose a valid position '*pos*' in $m_2$ randomly;
3: Transfer job $j_1$ to $m_2$ at position *pos*.

**Algorithm 6.** LS6:

1: Consider current solution $P_1$ and choose particle $P_2$ randomly;
2: Choose two crossover points in $P_1$.
3: All blocks outside the crossover points are inherited from $P_1$ to the same positions in new position ($N$) with no changes.
4: The genes that have already been selected from $P_1$ are deleted from $P_2$ so that the repetition of a gene in $N$ is avoided.
5: The other blocks in $N$ are sorted in the same order as in $P_2$ and complete the remaining empty gene locations with the undeleted genes that remain in $P_2$ by preserving their gene sequence.

**Algorithm 7.** LS7:

1: Consider current solution $P_1$ and randomly choose particle $P_2$;
2: Create a binary template (BT) and assign a randomly generated binary to each cell.
3: Copy the genes from $P_1$ corresponding to the locations of "1"s in the BT to the same positions in new position ($N$).
4: In order to avoid repetition of a gene in $N$, the copied genes to $N$ are deleted from $P_2$.
5: Complete the remaining empty gene locations with the undeleted genes remained in $P_2$ by preserving their genes sequence.

**Algorithm 8.** LS8:

1: Consider current solution $P_1$ and choose its P-best $P_2$;
2: Choose a crossover point in $P_1$.
3: All blocks in the right-hand side of crossover point are inherited from $P_1$ to the same positions in the new position ($N$) with no changes.
4: The genes that have already been selected from $P_1$ are deleted from $P_2$, so that the repetition of a gene in N is avoided.
5: The other blocks in $N$ are sorted in the same order as in $P_2$ and complete the remaining empty gene locations with the undeleted genes that remain in $P_2$ by preserving their gene sequence.

## 5. Computational Results

We have thoroughly reviewed the literature, and according to this review, the most related ones are APSO algorithm proposed by Ahmadi-Javid (2011), PSO algorithm proposed by Tseng and Liao (2008), and variable neighborhood search (VNS) proposed by Li et al. (2014). So, based on literature, we compare DPSO algorithm against these algorithms. Note that all algorithms are coded in C++.

### *5.1. Data generation and settings*

The problem data can be characterized by four factors, and each of these factors can have at least two levels. These levels are shown in Table 1.

Table 1
Factor levels

| Factor | Levels | | |
|---|---|---|---|
| Number of jobs | 6 | 30 | 100 |
| Machine distribution | Constant:1<br>Variable: Uniform(1, 4) | 2<br>Uniform(1, 10) | 10 |
| Number of stages | 2 | 4 | 8 |
| Processing times | Uniform(50, 70) | Uniform(20, 100) | |

The setup times are uniformly distributed from 12 to 24 which are 20% to 40% of the mean of the processing time (Rios-Mercado and Bard, 1998). The setup time matrices are asymmetric and satisfy the triangle inequality. Probability of skipping a stage is an important characteristic in the hybrid flowshop problem which, in this paper, is set at 0, 0.05, or 0.40 (Leon and Ramamoorthy, 1997). Therefore, there are 252 test scenarios. Note that some further restrictions are introduced. The largest number of machines in a stage must be less than the number of jobs. Thus, the combination with 10 machines at each stage and 6 jobs will be skipped and the combination of 1–10 machines per stage with 6 jobs will be changed to 1–6 machines per stage with 6 jobs. Also, the variable machine distribution factor requires that at least one stage have a different number of machines than the others.

*5.2. Stopping rules*

For fair comparison between algorithms, similar to Ruiz and Stützle (2008), we allocated equal time to algorithms. Furthermore, due to several runs, we are experimentally aware of the idea that computational time has direct relation with the number of jobs, machines, and stages. So, the stopping criterion is set to a computation time that is fixed to $(n^2 \times \sum_{t=1}^{g} m^t / g) \times 4$ milliseconds for all algorithms.

*5.3. Fine-tuning of the PSO algorithm parameters*

Fine-tuning the parameters of the metaheuristic algorithm is almost always a difficult task. The parameter values are extremely important because the results of the metaheuristic for each problem are very sensitive to them. This section describes an empirical testing approach to find the best tuning parameters of our proposed algorithm. We have applied parameters tuning for the population size (popsize), number of local search for G-best (LS), and selection mechanism in updating the position (UP) considering the following ranges:

- Population size: three levels (10, 20 and 50),
- Number of local searches for G-best: three levels (1, 2 and 3)
- The combination rule for each particle: three levels (elitism, crossover among three results, and serial implementation).

Twenty-seven different combinations are obtained by these levels. We generate six instances; two small, two medium and two large, for each combination of problem's factor levels. All these instances are solved by 54 different combinations.
The results are analyzed by the means of multi-factor analysis of variance (ANOVA) technique. It is necessary to notice that to use ANOVA, three main hypotheses, including normality, homogeneity of variance, and independence of residuals, must be checked. We did that and found no bias to question the validity of the experiment. Table 2 shows the results for different sizes of problems: small, medium, and large.

Table 2
Parameters tuning

| Parameters | Problems | | |
|---|---|---|---|
| | Small | Medium | Large |
| popsize | 20 | 20 | 50 |
| LS | 3 | 3 | 3 |
| UP | Serially | Serially | Serially |

*5.4. Evaluation metrics*

After computation of the objective value of each algorithm for its instances, the best solution obtained for each instance (which is named $Min_{sol}$) by any of the four algorithms is calculated. Relative percentage deviation (RPD) is obtained by the following formula:

$$RPD = \frac{\mathrm{Alg}_{sol} - Min_{sol}}{Min_{sol}} \quad (6)$$

where $A\lg_{sol}$ is the objective value obtained for a given algorithm and instance. RPD of 4% for a given algorithm means that this algorithm is 4% over the best obtained solution on average. Clearly, lower values of the RPD are preferred.

*5.5. Results*

*5.5.1. Analysis of makespan on RPD*

The results of the experiments are shown in Tables 3 and 4. As it can be seen, DPSO algorithm provides better results than the PSO, VNS, and APSO algorithms.

Table 3
Average relative percentage deviation ( $\overline{RPD}$ ) for algorithms grouped by n and g

| **Problem** | **Algorithm** | | | |
|---|---|---|---|---|
| | **VNS** | **APSO** | **PSO** | **DPSO** |
| **6×2** | 0.12705 | 0.06312 | 0.11021 | 0.10564 |
| **6×4** | 0.11557 | 0.08094 | 0.10732 | 0.02381 |
| **6×8** | 0.13760 | 0.06725 | 0.14472 | 0.01656 |
| **6 Job** | 0.12674 | 0.07043 | 0.12075 | 0.04867 |
| **30×2** | 0.06782 | 0.05151 | 0.06463 | 0.01143 |
| **30×4** | 0.03856 | 0.02589 | 0.04577 | 0.01024 |
| **30×8** | 0.03237 | 0.11498 | 0.12138 | 0.01048 |
| **30 Job** | 0.04625 | 0.06413 | 0.07726 | 0.01072 |
| **100×2** | 0.03053 | 0.01526 | 0.07728 | 0.00056 |
| **100×4** | 0.03150 | 0.02771 | 0.08696 | 0.00145 |
| **100×8** | 0.02524 | 0.01199 | 0.08084 | 0.00261 |
| **100 Job** | 0.02909 | 0.01832 | 0.08170 | 0.00154 |
| **Average** | 0.06736 | 0.05096 | 0.09323 | 0.02031 |

Table 4
Detailed results of algorithms

| **Problem size** | | **Agorithm** | **Results** | | |
|---|---|---|---|---|---|
| **Jobs** | **Stages** | | **Var** | **Min** | **Max** |
| **30 job** | **2 stage** | **APSO** | 25.38 | 2328.4 | 2341.4 |
| | | **VNS** | 46.95 | 2338.4 | 2357.2 |
| | | **PSO** | 118.93 | 2341.2 | 2367.2 |
| | | **DPSO** | 536.71 | 2334.7 | 2398 |
| | **4 stage** | **APSO** | 1986.23 | 1326.3 | 1443.2 |
| | | **VNS** | 32.76 | 1386.6 | 1402 |
| | | **PSO** | 50.05 | 1386.5 | 1404.1 |
| | | **DPSO** | 329.34 | 1274.8 | 1323.6 |
| | **8 stage** | **PSO** | 1547.99 | 830.82 | 929.57 |
| | | **VNS** | 64.97 | 808.06 | 827.95 |
| | | **SA** | 99.42 | 807.27 | 837.84 |
| | | **DPSO** | 2.27 | 789.55 | 793.96 |
| **100 job** | **2 stage** | **APSO** | 225.52 | 875.31 | 913.72 |
| | | **VNS** | 215.15 | 875.31 | 917.93 |
| | | **PSO** | 68.17 | 898.25 | 922.33 |
| | | **DPSO** | 7.06 | 858.46 | 865.96 |
| | **4 stage** | **APSO** | 264.96 | 4097.1 | 4142.6 |
| | | **VNS** | 207.23 | 4105.3 | 4142.6 |
| | | **PSO** | 191.86 | 4106.2 | 4140 |
| | | **DPSO** | 19.87 | 4104.1 | 4115.7 |
| | **8 stage** | **APSO** | 14677.71 | 3862.4 | 4186.2 |
| | | **VNS** | 51188.63 | 3644.4 | 4214.8 |
| | | **PSO** | 528.40 | 4164 | 4220.3 |
| | | **DPSO** | 1595.27 | 3906.4 | 4011.4 |

Var: Variance
Min: Minimum
Max: Maximum

In order to verify the statistical validity of the results shown in Table 3 and to confirm which the best algorithm is, we performed a design of experiments and an analysis of variance (ANOVA) in which the algorithms are the factor and RPDs are the response variable. The means plot and LSD intervals (at the 95% confidence level) for four algorithms are shown in Figure 3.

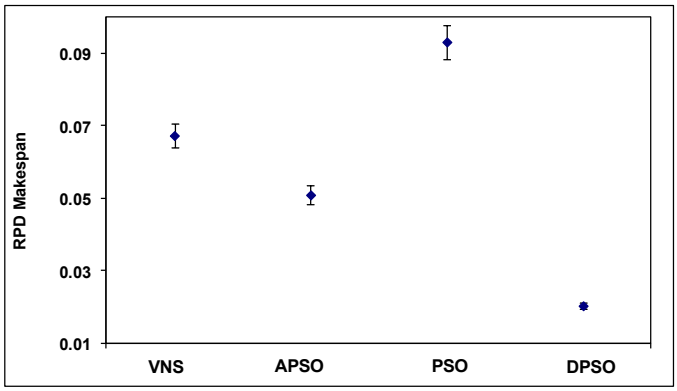


Fig. 3. Plot of $\overline{RPD}$ for the type of algorithm factor

The results demonstrate that there is a clear statistically significant difference among the performances of the algorithms.

The convergence plot of the proposed algorithm for three large-sized samples is also presented in Figure 4.

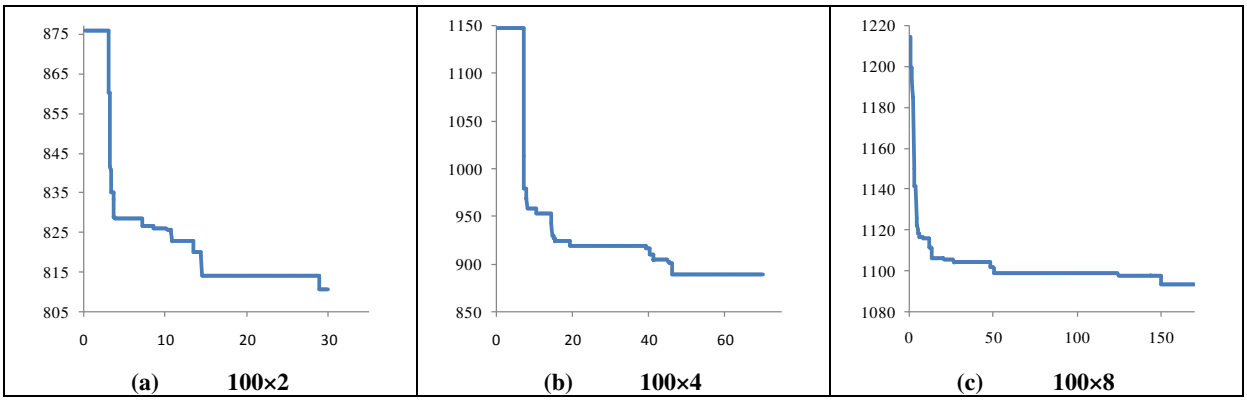


Fig. 4. Convergence plot of the proposed algorithm for three large-sized instances

*5.5.2. Analysis of controlled factors*

ANOVA-F test was carried out to determine whether the treatment means are significantly different from each other. All tests were conducted at the 5% level of significance.

*5.5.2.1. Analysis of problem size factor (number of jobs)*

In order to see the effects of number of jobs on four algorithms, a two-factor ANOVA is applied. Plot of $\overline{RPD}$ for the interaction between the type of algorithm and number of jobs is shown in Figure 5.

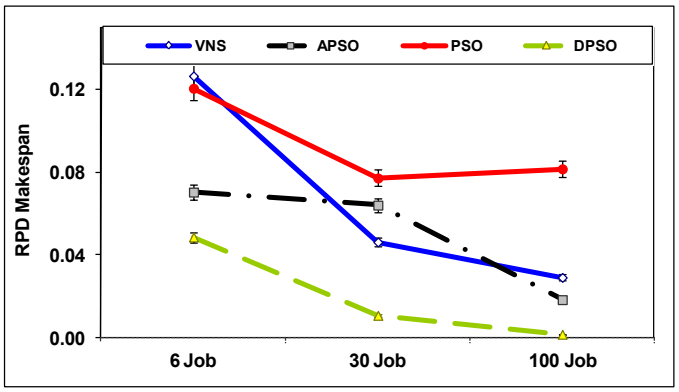


Fig. 5. Plot of $\overline{RPD}$ for the interaction between the type of algorithm and the number of jobs

As we can see, in all cases, the DPSO algorithm works better than PSO, VNS, and APSO algorithms.

*5.5.2.2. Analyzing factor g (number of stages)*

Other two-factor ANOVA and LSD tests are applied to see the effect of magnitude of stages on quality of the algorithms. The results are shown in Figure 6.

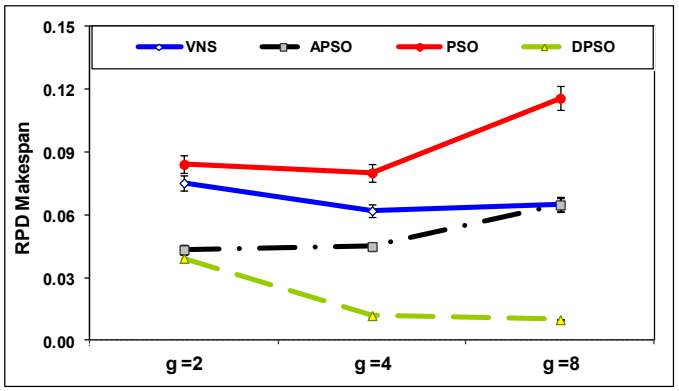


Fig. 6. Plot of $\overline{RPD}$ for the interaction between the type of algorithm and magnitude of the stages

As we can see, in the cases of $g = 2$, $g = 4$, and $g = 8$, the DPSO algorithm works better than others.

## 6. Conclusions and Future Works

Particle swarm optimization as a well-known metaheuristic has desirable properties, but the convergence usually happens prematurely and parts of solution space remain unexplored. Moreover, the standard particle swarm optimization (PSO) algorithm cannot be applied to discrete problems. To overcome these shortcomings, in this paper, we introduce diversified particle swarm optimization (DPSO) algorithm as a powerful generalization of PSO algorithm. DPSO algorithm is inspired by the concept of a society whose members behave anarchically to improve their situations. In DPSO algorithm, the particles are fickle, and their fickleness increases as their situations become worse. They also occasionally behave irregularly and move towards worse particles of the swarm or worse positions which they have previously visited. DPSO algorithm of such anarchic particles explores the solution space perfectly and prevents falling in the local optimum traps. Recently, sequence-dependent hybrid flowshop (SDST-HFSP) has been solved by almost all the well-known heuristic methods. We numerically compare the performance of the DPSO algorithm with three of the best algorithms recently developed in the literature. A comprehensive set of computational experiments and statistical analyses for test instances with different structures has been carried out. To fairly compare the algorithms, we allocated equal times to all the algorithms which was also enough for all of them to converge. For all the combinations of the problem parameters, especially for medium and large instances, the DPSO algorithm considerably outperforms the other ones. According to our promising results, we expect that DPSO algorithm can be successfully used in solving other challenging discrete problems. Also, since the proposed DPSO algorithm framework includes the standard PSO algorithm which is

widely used for continuous problems, applying DPSO algorithm to continuous problems is another interesting research area.

**References**


Ahmadi-Javid, A. (2011). Anarchic Society Optimization: A Human-Inspired Method. *In 2011 IEEE Congress on Evolutionary Computation (CEC)*, 2586-2592.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs, *European Journal of Operational Research*, 377(2), 345-378.

Behnamian, J. Fatemi Ghomi, S.M.T., & Zandieh, M. (2009). A multi-phase covering Pareto-optimal front method to multi-objective scheduling in a realistic hybrid flowshop using a hybrid metaheuristic. *Expert Systems with Applications,* 36(8), 11057-11069.

Behnamian, J. Fatemi Ghomi, S.M.T., & Zandieh, M. (2012). Hybrid flowshop scheduling with sequence-dependent setup times by hybridizing max–min ant system, simulated annealing and variable neighborhood search, Expert *Systems: The Journal of Knowledge Engineering*, 29 (2), 156–169.

Blackwell, T., & Bentley, P.J. (2002). Don't push me! Collision-avoiding swarms. In*: Proceedings of the IEEE congress on evolutionary computation*, 1691–96.

Chen, C-C. (2011). Two-layer particle swarm optimization for unconstrained optimization problems, *Applied Soft Computing*, 11(1), 295–304.

Clerc, M. (2006). Particle swarm optimization. ISTE;

Coelho, L.S. (2008). A quantum particle swarm optimizer with chaotic mutation operator, *Chaos, Solitons and Fractals*, 37, 1409–1418.

Coelho, L.S. (2009). Reliability–redundancy optimization by means of a chaotic differential evolution approach , *Chaos, Solitons & Fractals*, 41(2), 594-602

Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. *In: Proceedings of the IEEE Sixth International Symposium on Micro Machine and Human Science*, 39-43.

El-Abd, M. Hassan, H. Anis, M. Kamel, M.S., & Elmasry, M. (2010). Discrete cooperative particle swarm optimization for FPGA placement. *Applied Soft Computing*, 284-295.

Gaafar, L.K. Masoud, S.A., & Nassef, A.O. (2008). A particle swarm-based genetic algorithm for scheduling in an agile environment. *Computers & Industrial Engineering*, 55, 707–720.

García-Villoria, A., & Pastor, R. (2009). Introducing dynamic diversity into a discrete particle swarm optimization, *Computers & Operations Research*, 36(3), 951-966.

He, S. Wu, Q.H. Wen, J.Y. Saunders, J.R. R.C., & Paton, (2004). A particle swarm optimizer with passive congregation. *Biosystems*, 78, 135–47.

Jie, J. Zeng, J. Han, C., & Wang, Q. (2008). Knowledge-based cooperative particle swarm optimization. *Applied Mathematics and Computation,* 205 (2), 861-873.

Jina, Z. Yang, Z., & Ito, T. (2006). Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. *International Journal of Production Economics*, 100(2), 322-334.

Johnson, D.S. Aragon, C.R. Mcgeoch, L.A., & Schevon, C. (1989). Optimization by simulated annealing: an experimental evaluation; Part I, graph partitioning, *Operations Research*, 37(6), 865–892.

Karthi, R. Arumugam, S., & Ramesh Kumar, K. (2009). Discrete Particle Swarm Optimization Algorithm for Data Clustering Nature Inspired Cooperative Strategies for Optimization (NICSO 2008).

Koua, X. Liu, S. Zhang, J., & Zheng, W. (2009). Co-evolutionary particle swarm optimization to solve constrained optimization problems. *Computers & Mathematics with Applications*, 57, 11-12.

Król, D., & Drożdżowski, M. (2010). Use of MaSE methodology and swarm-based metaheuristics to solve the traveling salesman problem. *Journal of Intelligent and Fuzzy Systems*, 21(3), 221-231.

Kurz, M.E., & Askin, R.G. (2003). Comparing scheduling rules for flexible flow lines. *International Journal of Production Economics*, 85, 371-388.

Kurz, M.E., & Askin, R.G. (2004). Scheduling flexible flow lines with sequence-dependent setup times. *European Journal of Operational Research*, 159, 66–82.

Laskari, E.C. Parsopoulos, K.E., & Vrahatis, M.N. (2002). Particle swarm optimization for integer programming. *In: Proceedings of the IEEE 2002 Congress on Evolutionary Computation, Honolulu (HI),* 1582–1587.

Leon, V.J., & Ramamoorthy, B. (1997). An adaptable problem-space based search method for flexible flow line scheduling. *IIE Transactions*, 29, 115–125.

Li, J-Q., Pan, Q-K. , & Wang, F-T. (2014). A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem. *Applied Soft Computing,* (24), 63–77.

Lozvbjerg, M. Krink, T. (2002). Extending particle swarms with self-organized criticality. *In: Proceedings of the IEEE congress on evolutionary computation*,1588–93.

Montalvo, I. Izquierdo, J. Pérez, R., & Tung, M.M. (2008). Particle swarm optimization applied to the design of water supply systems. *Computers & Mathematics with Applications*, 56(3), 769-776.

Naderi, B. Zandieh, M., & Aminnayeri, M. (2011). Incorporating periodic preventive maintenance into flexible flowshop scheduling problems. *Applied Soft Computing*, 11(2), 2094–2101.

Nawaz, M. Enscore, E., & Ham, I. (1983). A heuristic algorithm for the *m*-machine, n-job flow-shop sequencing problem. *Omega*, 11, 91–95.

Parsopoulos, K.E. Vrahatis, D.K., & Tasoulis, M.N. (2004). Multi-objective optimization using parallel vector evaluated particle swarm optimization. *In: Proceedings of the IASTED international conference*

*on artificial intelligence and applications,* 2, 823–828.

Parsopoulos, K.E., & Vrahatis, M.N. (2001). Particle swarm optimizer in noisy and continuously changing environments. *In: Hamza MH, editor, Artificial intelligence and soft computing*, 289–94.

Pinedo, M.L. (2012). Scheduling Theory, Algorithms, and Systems, Fourth Edition, *Springer*, NY.

Rios-Mercado, R.Z., & Bard, J.F. (1998). Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. *Computers & Operations Research*, 25 (5), 351–366.

Ruiz, R., & Stützle, T. (2008). An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3), 1143–1159.

Sadati, N. Amraee, T., & Ranjbar, A.M. (2009). A global particle swarm- based-simulated annealing technique for undervoltage load shedding problem. *Applied Soft Computing*, 9(2), 652–657.

Sha, D.Y., & Hs, C-Y. (2006). A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering*, 51(4), 791-808

Sun, T-H. (2009). Applying particle swarm optimization algorithm to roundness measurement. *Expert Systems with Applications*, 36 (2), 3428-3438.

Talbi, El-G. (2009). Metaheuristics: From Design to Implementation, *Wiley Series on Parallel and Distributed Computing*.

Tang, Y. Qiao, L., & Guan, X. (2010). Identification of Wiener model using step signals and particle swarm optimization. *Expert Systems with Applications*, 37(4), 3398-3404.

Tseng, C-T., & Liao, C-J. (2008). A particle swarm optimization algorithm for hybrid flow-shop scheduling with multiprocessor tasks. *International Journal Of Production Research*, 46(17), 4655-4670.

Wang, J. Kuang, Z. Xu, X., & Zhou, Y. (2009). Discrete particle swarm optimization based on estimation of distribution for polygonal approximation problems. *Expert Systems with Applications*, 36 (5), 9398-9408.

Wang, X., & Tang, L. (2009). A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers. *Computers & Operations Research*, 36(3), 907–918.

Wang, Y., & Liu, J.H. (2010). Chaotic particle swarm optimization for assembly sequence planning. *Robotics and Computer-Integrated Manufacturing*, 26(2), 212-222.

Xiang, T. Wong, K-w., & Liao, X. (2007). A novel particle swarm optimizer with time-delay. *Applied Mathematics and Computation*, 186 (1), 789-793.

Xie, X.F. Zhang, W.J., & Yang, Z.L. (2002). A dissipative particle swarm optimization. *In: IEEE congress on evolutionary computation (CEC'02)*, HI, USA.

Xiong, Y. Cheng, H-Z. Yan, J-Y., & Zhang, L. (2007). New discrete method for particle swarm optimization and its application in transmission network expansion planning. *Electric Power Systems Research*, 77(3-4), 227-233.

Yang, Y. Xiaoxing, L., & Chunqin, G. (2008). Hybrid particle swarm optimization for multiobjective resource allocation, *Journal of Systems Engineering and Electronics*, 19(5), 959-964.

Yeh, W-C. Chang, W-W., & Ying Chung, Y. (2009). A new hybrid approach for mining breast cancer pattern using discrete particle swarm optimization and statistical method. *Expert Systems with Applications*, 36(4), 8204-8211.

Yin, P-Y. (2004). A discrete particle swarm algorithm for optimal polygonal approximation of digital curves, *Journal of Visual Communication and Image Representation*, 15(2), 241-260.

**This article can be cited:** Behnamian, J. (2019) Diversified Particle Swarm Optimization for Hybrid Flowshop Scheduling. *Journal of Optimization in Industrial Engineering*. 12 (2), 107-119.

http://www.qjie.ir/article_538342.html
DOI: 10.22094/JOIE.2018.671.1433