

## Particle Swarm Optimization and Tabu Search Hybrid Algorithm for Flexible Job Shop Scheduling Problem – Analysis of Test Results

*Asen Toshev*

*Institute of Information and Communication Technologies, BAS, 1113 Sofia, Bulgaria  
E-mail: tochevassen@yahoo.com*

**Abstract:** *The paper presents a hybrid metaheuristic algorithm, including a Particle Swarm Optimization (PSO) procedure and elements of Tabu Search (TS) metaheuristic. The novel algorithm is designed to solve Flexible Job Shop Scheduling Problems (FJSSP). Twelve benchmark test examples from different reference sources are experimentally tested to demonstrate the performance of the algorithm. The obtained mean error for the deviation from optimality is 0.044%. The obtained test results are compared to the results in the reference sources and to the results by a genetic algorithm. The comparison illustrates the good performance of the proposed algorithm. Investigations on the base of test examples with a larger dimension will be carried out with the aim of further improvement of the algorithm and the quality of the test results.*

**Keywords:** *Flexible Job Shop Scheduling Problem (FJSSP), Particle Swarm Optimization (PSO), Tabu Search (TS).*

### 1. Introduction

#### 1.1. Formulation of Job Shop Scheduling Problem (JSSP)

The classical [44] job shop scheduling problem is formulated as follows: Let us have a set of  $n$  jobs  $J_1, \dots, J_n$ , which have to be performed on  $m$  machines  $M_1, \dots, M_m$ .

The operations consequence is given for each job:

$J_i = (O_{i1}, \dots, O_{ij(i)}), j(i)$  is the number of operations for the corresponding job,  $i = 1, \dots, n$ .

It strictly determines which operation on which machine should be executed.

For this model another formulation is:

$$J_i = (M_{i1}, \dots, M_{in}),$$

The processing times for each possible operation are known for each machine:

$$p_{ik}, i = 1, \dots, j(i), k = 1, \dots, m.$$

The optimal schedule should be found according preliminary given criterion (criteria). One criterion could be the minimization of makespan –  $C_{\max}$  for example, where  $C_{\max}$  corresponds to the maximal time necessary for completion all jobs.

This is chronologically the earliest developed model and the most often used – see for example [10, 21, 18].

## 1.2. Formulation of Flexible Job Shop Scheduling Problem (FJSSP)

An extension of classical JSSP taking into account the production flexibility is the FJSSP. Each operation  $O_{ij}$  in the FJSSP can be processed on one machine  $M_k$  from a set of several machines  $M_k \in M_{ij}$ ,  $M_{ij} \subseteq M$ , unlike the classical JSSP where each operation is processed on a predefined machine. We have Partial Flexibility of JSSP (P-FJSSP), if  $M_{ij} \subset M$  for at least one operation; then we have Total Flexibility of JSSP (T-FJSSP), if  $M_{ij} \equiv M$  for each operation.

This model corresponds to real life production situations [37], where the machines could perform more than one operation (not at the same time) with corresponding different production times, i.e., some or all of the machines are multifunctional. Bruker and Schlie [8], are among the first researchers suggesting this model.

The FJSSP is NP-hard [10, 22, 30].

If the number of jobs is 2, or if the number of machines is 2 and all jobs have 1 or 2 operations, or if the number of machines is 2 and all operations have duration 1, then the optimal solutions for job shop scheduling can be found in polynomial time. In all other cases the FJSSP is NP-hard – [10, 22], i.e., if the problem is obtained by incrementing the number of machines, jobs, operations or durations by 1.

Lists of special polynomially solvable JSSP cases and flexible JSSPs, as well as their simplest NP-hard common versions, can be found in [7] and the references inside. The author focuses on JSSP and flexible JSSP. Challenges, benchmark tasks, exact algorithms to solve them are listed in chronological order, and some surveys are mentioned. Author concludes that examples with a size greater than  $15 \times 15$  should be solved through heuristics and offers heuristics with Tabu Search (TS) as the best known metaheuristic for JSSP. Real applications are listed to lead to advanced JSSP models and concepts for extending the tasks associated with these applications are presented.

Hybrid heuristics have become very popular due to their efficiency. They are often based on the use of local search in evolutionary approaches. Such methods are sometimes called “memetic algorithms”. For example, they include local search by Vaessens, Aarts and Lenstra [40], the critical point shift approach by Adams, Balas and Zawack [1], making local search with critical points shifting approach by Balas and Vazacopoulos [3], the limited distribution algorithm by Brinkkötter and Brucker [6], the parallel greedy search procedure using random number generation (GRASP) by Aiex, Binato and Resende [2] and procedures similar to the mentioned.

In the last few years, the Particle Swarm Optimization (PSO) has been successfully applied in many areas of research and applications. It has been demonstrated that PSO generates better results (mainly in terms of convergence of

the search process) in a faster and less expensive way than other evolutionary methods.

Xia and Wu [41] offer a hybrid algorithm combining swarm optimization and simulated annealing to solve the task of a flexible production schedule.

Rahimi-Vahed and Mirghorbani [32] developed a multicriterial swarm optimization algorithm designed to minimize both the weighted average completion time and the weighted average delay in a task for flow shop schedules. They conclude that for large-scale tasks, the algorithm developed is more effective than a genetic algorithm for the same task. A reference point genetic algorithm for multi-criteria FJSSP with good efficiency is proposed by Gulia shki and Kirilov [16].

Zhang et al. [42] offer a hybrid algorithm combining swarm optimization and taboo search technology to solve the task of a flexible production schedule.

The original PSO procedure is typically used to solve optimization tasks with continuous variables. When discrete decision space is used in the job shop scheduling problem, the representation of particles, particle motion, and particle velocity should be modified.

Sha and Lin [36] construct a Particle Swarm Optimization Algorithm (PSO Algorithm) for multicriteria JSSP. Criteria used in this case are: the makespan, the total delay, and the total standby time of the machines.

Since the best known heuristic algorithms for JSSP and FJSSP are based on TS and PSO naturally arises the idea for a development of a combined PSO&TS technique for this class of optimization problems. A new hybrid PSO&TS algorithm is presented in this paper and its performance is illustrated on a sample of 12 benchmark test examples.

The paper is organized as follows. In Section 2 a PSO-based sub-procedure is presented. Section 3 is devoted to Tabu Search metaheuristic and some useful TS-elements which are included in the hybrid algorithm. The new hybrid algorithm developed is presented in Section 4. Test results and their analysis are considered in Section 5. Some conclusions are drawn in Section 6. Directions for further research are outlined.

## 2. Particle Swarm Optimization (PSO)

PSO is based on a population-based stochastic optimization technique that is produced by the movement of bird flocks, passages of fishes and swarms of insects. PSO has many common features with evolutionary optimization techniques such as Genetic Algorithms (GA) [25]. The system is initialized with a random population and looking for optimum by renewing generations. Unlike GA, PSOs have no evolutionary operators such as crossovers and mutations. In PSOs Ramasewych and Lovelkin [33], potential solutions, called particles, fly in the feasible domain of the problem, following the current optimal particle. Each particle saves its coordinates in the feasible domain. Also the coordinates of the best solution (with best fitness value) achieved so far are saved. This value is called pbest. Another "best" value used in the particle swarm optimization is the best value obtained so far

by each particle in the neighborhood of the concrete particle. This location is denoted by lbest. If the whole population is considered as a neighborhood of the particle, its best value is the global one and it is called gbest.

The concept of particle swarm optimization [34] consists in changing at each time step the velocity of every particle to its pbest and lbest (the local version of the PSO) locations. When determining the final step to the new localization, the particle inertia is also involved, i.e., its current speed is also taken into account. In PSO, each solution is one point in the search space and it can be considered as one particle. The original set of particles is generated randomly in the feasible domain/the search space. In their movement the particles have memory and each particle regulates its position based on its experience, as well as on the experience of the particles gbest and lbest. The previous best position of the particle is denoted as pbest (local best). At each iteration, the velocity and the position of each particle should be updated on the base of lbest and gbest values.

Suppose the space of the search is  $d$  and the position of the  $i$ -th particle in the  $t$ -th iteration as  $X_i^t = (x_{i1}^t, x_{i2}^t, \dots, x_{id}^t)$  is shown. Velocity of the  $i$ -th particle of the  $t$ -th iteration is  $V_i^t = (v_{i1}^t, v_{i2}^t, \dots, v_{id}^t)$ . The best position of the  $i$ -th particle on the  $t$ -th is  $P_i^t = (p_{i1}^t, p_{i2}^t, \dots, p_{id}^t)$ . The best position of the whole swarm is  $G^t = (g_1^t, g_2^t, \dots, g_d^t)$  on the  $t$ -th iteration. The rapidity and the position of the whits in the standard PSO should change with the equations

$$(1) \quad v_i^{t+1} = \omega \times v_i^t + c_1 \times \text{rand}() \times (p_i^t - x_i^t) + c_2 \times \text{rand}() \times (g_i^t - x_{id}^t),$$

$$(2) \quad x_i^{t+1} = x_i^t + v_i^{t+1},$$

where:  $x_i^t$  and  $v_i^t$  are the position and the velocity of the  $n$ -th component of one particle on the  $n$ -th iteration, respectively.  $\omega$  is an inert burden designed to balance global and local capabilities, and  $\text{rand}()$  is randomly chosen from the normal distribution in the range  $[0, 1]$ .  $c_1$  and  $c_2$  are acceleration constants that must limit the rate of acceleration that applies to all parts of pbest and gbest, respectively. The changing shape strengthens the particles that move toward the clustering vector of local and global optimal solutions. That is why the particles' possibility to reach an optimal solution is growing.

The optimization through swarm particles have some advantages and disadvantages.

1) Advantages are:

- Swarm optimization is based on intelligence. It can be used for research and engineering.
- Swarm particle optimization has no overlapping and mutational computation. The test is carried out by the particle velocity. During the development of different generations, only the most optimistic particle can exchange information with other particles, so the speed of exploration is very high.
- Calculation in optimization by swarm particles is very simple. Compared to other developed calculations, it has greater optimization capability and can be easily completed.

- Optimization through a swarm of particles adapts a real number code, and it is solved directly through the solution. The magnitude of the dimension is directly determined by the decision constant.

2) Disadvantages are:

- The method is easily dependent on partial optimism, which causes less accuracy of its speed and direction regulation.
- The method cannot have a solution for spreading and optimization.
- The method cannot have a solution when there is no coordinate system as a solution in an energy field and any movement of the particle rules in an energy field.

**The Current state of the optimization (algorithm) by swarm particles.** The swarm optimization is based on the intelligence of swarm particles. The research is still in its early stages. Unlike genetic algorithms and simulated annealing, the metaheuristic have not still a mathematical basis and a systematic computational method.

The mathematical foundation includes the mechanics of movement itself, the proof of their convergence, the resistance, and so on.

Applied research involves the continuation of its advantages, the overcoming of its disadvantages and the development of its scope.

Research on PSOs should focus on the following: 1) Put some new technologies in the PSO; 2) PSO can be combined with other optimization techniques to describe optimization problems and solve them; 3) PSO can be led to run the research in spraying systems, optimistic systems, and non-coordinate systems to develop applied PSO fields.

Metaheuristic optimization through swarm particles generally should contain the following elements: an initial solution, the best personal (particle) solution, the best global (swarm) solution, a formula on which to calculate particle speed and position, braking criteria.

Improving meta-optimization through swarm particles must include:

- 1) The introduction of inert weights;
- 2) Increasing the convergence factor;
- 3) Entering the selection;
- 4) The basic process in swarm particle optimization depends on the basic algorithm set in the PSO.

The future development of PSOs must contain:

- 1) Development of the mathematical foundations of the algorithm;
- 2) Development of the topography of the swarm particles;
- 3) Combining PSO with other metaheuristics using the good sides of each;
- 4) Extension of the field of metaheuristics application.

#### **Algorithm for metaheuristic optimization through swarm particles**

For each particle  $i = 1, \dots, S$  make:

Initialize the position of the particle with a random vector:  $x_i \sim t_{ij}^k$ , where  $t_{ij}^k$  is randomly selected process time of operation  $k$  for machine  $i$  and operation  $j$  in the search space.

Initialize the best known position of the particle with its initial position:  $p_i \leftarrow x_i$

If  $(f(p_i) < f(g))$  update the best known position of the swarm:  $g \leftarrow p_i$

Initialize particle velocity using the formula:

$$(3) \quad v_{id} \sim \text{int} [\omega v_{id} + c_1 r_1 \text{dis}(p_{id} - x_{id}) + c_2 r_2 \text{dis}(p_{gd} - x_{id})].$$

As long as the end criterion meets (the number of iterations performed is reached or an adequate target function value is found), repeat:

For each particle  $i = 1, \dots, S$  make:

For each dimension  $d = 1, \dots, n$ , make:

Generate random numbers  $r_1, r_2 \sim U(0, 1)$

Update the particle velocity with:

$$(4) \quad v_{id} \leftarrow \text{int} [\omega v_{id} + c_1 r_1 \text{dis}(p_{id} - x_{id}) + c_2 r_2 \text{dis}(p_{gd} - x_{id})],$$

update the position of the particle:  $x_i \leftarrow x_i + v_i$

(The operation to be moved, we assign it to the first position.

The current operation we are looking at is put on the first free machine than can run it for the least time.)

If  $(f(x_i) < f(p))$  do:

Upgrade the best known particle position:  $p_i \leftarrow x_i$

If  $(f(p_i) < f(g))$ , update the best known position of the swarm:  $g \leftarrow p_i$ .

Now  $g$  contains the best solution found.

#### **Algorithm for binary optimization by swarm particles for a flexible production schedule**

**i.** Finding initial solutions  $X_h^0$ ,  $h = 1, 2, \dots, N_p$ , where  $N_p$  is the number of particles in the population.

The initial population is calculated as follows: the current schedule is compiled on the basis of generated random numbers  $r \in (0, 1)$ . The interval  $(0, 1)$  is divided into  $n$  parts corresponding to each job. Each random number defines the current work from which an operation is taken to be included in the schedule. Taking the current operation from the selected operation, the strict order of operations is strictly observed. Operations are set on machines according to the rule for the first free machine, and if there are several free machines at the same time, the machine with the smallest index is selected. In case the current job is completed (i.e., all of its operations are included in the graph), it continues with unfulfilled work that has the smallest index until the works are exhausted.

Let  $O$  be the number of operations for the task under consideration. Follow the following algorithm:

If  $1 \leq O \leq 25$ , then  $N_p = 5$ ;

If  $25 < O \leq 50$ , then  $N_p = 10$ ;

If  $50 < O \leq 75$ , then  $N_p = 15$ ;

If  $75 < O \leq 100$ , then  $N_p = 20$ ;

If  $100 < O$ , then  $N_p = 30$ .

**ii.** Applying at the start moment of the particle velocity  $V_{hij}^0 = 0$ ,  $i = 1, 2, \dots, O$  ( $i$  is the operation number);  $j = 1, 2, \dots, M$ , where  $M$  is the number of machines, ( $j$  is the machine number).  $X_{hij}^0$  and  $V_{hij}^0$  are the elements of  $X_h^0$  and  $V_h^0$ , respectively.

**iii.** Finding the binary matrix  $B_{hij}^0$ ,  $B_{hij}^0 \in \{0, 1\}$ , where is unit 1, there  $X_{hij}^0$  has a number different from zero 0. In a particular timetable, an unit in  $B_{hij}^0$  can have

only one column on the  $i$ -th line, so only one machine can handle a given operation. The other places have zero 0.

iv. Finding the speed  $V_{hij}^k$  ( $h$  is the particle number in the population,  $i$  is the operation number,  $j$  is the machine number, and  $k$  is the current iteration) by the formula

$$(5) \quad V_{hij}^k = \omega * V_{hij}^{k-1} + r_1 * c_1 * \text{dist}(P_{hij}^{k-1}, X_{hij}^{k-1}) + r_2 * c_2 * \text{dist}(G_{hij}^{k-1}, X_{hij}^{k-1}),$$

where  $\omega$ ,  $c_1$ ,  $c_2$  are coefficients determined by the decision maker and they control the behavior of the particle PSO,  $\omega = 1$ ,  $c_1 = 1.8$ ,  $c_2 = 2.2$ ,  $r_1$  and  $r_2$  are two random numbers generated with an uniform distribution,  $r_1, r_2 \in [0, 1]$ ,  $P_h^k$  is the locally best position in the vicinity of the  $X_h^k$  particle, and  $G^k$  is the globally best position for the  $X$  population.

The distance is defined as follows:

$$(6) \quad \text{dist}(P_{hij}^k, X_{hij}^k) = t * [a * |f(P_{hij}^k) - f(X_{hij}^k)| / C + b * (D - S(P_{hij}^k, X_{hij}^k)) / D],$$

$t$  is a positive integer called accelerating coefficient,  $a$  and  $b$  are two positive weights that can be obtained experimentally and fulfill the condition that their sum equals 1. From Ge, Du and Qian [11],  $t = 1$  was taken,  $a = 0.7$  and  $b = 0.3$ ,  $f(X_{hij}^k)$  is a fitness function,  $0 \leq f(X_{hij}^k) \leq C$ , for the task being considered, this is the makespan.  $D$  is the size of the space, in this case  $D = O$  is the number of operations.

Let  $X_i(x_{i1}, x_{i2}, \dots, x_{iO})$  and  $X_j(x_{j1}, x_{j2}, \dots, x_{jO})$  be two particles in the space of operations. The following functions are entered:

$$(7) \quad s(m) = \begin{cases} 1, & x_{im} = x_{jm}, \\ 0, & x_{im} \neq x_{jm}, \end{cases}$$

and the function  $S(X_i, X_j)$ , which is called *function of similarity*.  $S(X_i, X_j) = \sum_{m=1}^O s(m)$ .

Only those  $V_{hij}^k$ 's that have 1 are calculated, so if it becomes 0 in a sigmoid function calculation, this is reflected in  $X_{hij}^k$ , with zero being written in the old position and a unit in the alternative position at  $M = 2$ ; and for  $M > 2$ , one of the following options is selected:

1. In the position of the smallest process time for this  $i$ -th order;
2. In the position of the longest processing time for this  $i$ -th order;
3. In the position of the closest to the average process time for this  $i$ -th order.

Let

$$(8) \quad \text{sigma}(V_{hij}^k) = 1 / (1 + \exp(-V_{hij}^k)),$$

is the sigmoid function for  $V_{hij}^k$ . In order to prevent the excessive approximation of  $\text{sigma}(V_{hij}^k)$  to 0 or 1, a constant  $V_{\max}$  is often taken to limit  $V_{hij}^k$ 's magnitude. Typically,  $V_{\max} = 4$ ;  $V_{hij}^k \in [-V_{\max}, V_{\max}]$ . After the transformation (8) the  $\text{sigma}(V_{hij}^k)$  is in the range of 0 to 1, i.e.,  $\text{sigma}(V_{hij}^k) \in (0, 1)$ .

Find the result of the change in position using the following rule using the probability function:

$$(9) \quad B_{hij}^k = \begin{cases} 1 & \text{if random} < \text{sigma}(V_{hij}^k); \\ 0 & \text{otherwise.} \end{cases}$$

v. Find the matrix  $B_{hij}^k$ ,  $h = 1, 2, \dots, N_p$ ,  $i = 1, 2, \dots, O$ ,  $j = 1, 2, \dots, M$  according (7);  $B_{hij}^k \in \{0, 1\}$ .

**vi.** Finding a new  $X_h^k$  solution according to the  $B_{hij}^k$  matrix, where there is 0, put 0, and where there is 1, place the process time.  
 If  $f(X_h^k) < f(P_h^{k-1})$ , then  $P_h^k = X_h^k$ , otherwise  $P_h^k = P_h^{k-1}$ ,  $h = 1, 2, \dots, N_p$ .  
 If  $f(X_h^k) < f(G_{k-1})$ , then  $G_k = P_h^k$ , else  $G_k = G_{k-1}$ ,  $h = 1, 2, \dots, N_p$ .  
 Let iter\_lim be the number of iterations that are pending for execution.  
 If the stop criterion is met, i.e.:  
 1) reached iter\_lim;  
 2) there is no improvement of the current iteration of the iteration;  
 3) the best makespan in the population is equal to the sum of the optimal times for each job  
 or  
 4) the value of the target function, as desired by the decision maker,  
 end,  
 otherwise it goes to **iv**.

### 3. Tabu search

This is one of the most widely used metaheuristics designed mainly to solve combinatorial optimization problems, like transport nets control, distribution of electro energy, schedules, etc. The main idea of this algorithm was first introduced by Glover [13, 14]. The basic algorithm includes a local search with the greatest improvement (best-fit) and a short term memory to avoid the local optima and the cycling. The short term memory is applied as a Tabu list, where the last solutions considered are stored and the movements directed towards them are forbidden. The neighborhood of a current solution includes only decisions, which are not in the Tabu list. The set of these solutions is called an allowed set. At each iteration, the best solution of this set is chosen as a new current solution. This solution is included in the Tabu list and one of the solutions stored in it is removed (usually FIFO order is used). In order to avoid the local optimum, movements towards worse neighbor solutions are allowed. Another type of memory is also used, called long term memory, where information about past search steps is stored, as well as how many times a given solution has been chosen and the frequency of changing one concrete solution, etc. This memory is used to direct the search to regions of the feasible domain, which have still remained unexplored, i.e., its purpose is to realize diversification of the search. Usually the search procedure is terminated after executing a given limit of iterations or after a given number of consecutive iterations without improving the best obtained solution.

The good performance of the TS approach depends mainly on the balance between the intensification and the diversification scheme.

TS can be applied to virtually any optimization problem [15]. We can present each of these issues in the following way, where “optimization” means maximization or minimization:



Optimize  $f(x)$ ,  $x \in X$ .

The function  $f(x)$  can be linear, nonlinear or stochastic, and the set  $X$  generates the constraints on the vector of the variable  $x$ . Similarly, the  $x$  constraints can be linear, non-linear or stochastic inequalities, and may force all or some of the  $x$  components to obtain discrete values.

According to Glover, Laguna and Marti [15] for the metaheuristic Tabu Search the following principles can be formulated:

1. Selectivity (including strategic forgetting).
2. Abstraction and decomposition (by explicit and attributive memory).
3. Time allocation:
  - a novelty of things;
  - frequency of things;
  - differentiation between short-term and long-term.
4. Quality and Influence:
  - relative attractiveness of alternative chances;
  - magnitude of change of structure or limiting ratios.
5. Context:
  - regional interdependence;
  - structural interdependence;
  - seamlessly interdependence.
6. Responsive Study.
7. Strategically used constraints and incentives (tabu conditions and aspiration levels).
8. Concentrated focus in good regions and good decisional features (intensification processes).
9. Characterization and exploration of new regions (diversification processes).
10. Non-monotonous Search Templates (Strategic Oscillation).
11. Integration and expansion solutions (connecting the paths).

TS is rapidly becoming [9] a method for describing decisive procedures for complex combinatorial tasks. Many practical successes of the method contribute to rapid dissemination in the sense of finding extremely high quality. TS methods are used to create hybrid procedures with other heuristics and algorithms [9], to offer timetables, resource allocation, investment planning, telecommunications and many other areas.

In general, the metaheuristic Tabu Search (TS) should contain the following elements [29]: Initialization solution, structure of neighbors, movements, taboo list, aspiration criteria, memory structures and end criteria.

**1. Initialization solution.** To initiate the TS process, there must be an appropriate initialization solution. It can be obtained, for example, by using dispatch rules. The good quality of the initial decision often has a decisive impact on the process of seeking an optimal solution. Policies are often used, such as the least process time, the longest running time, the closest to the average process time, and so on.

In this paper, an initial solution of the Tabu Search algorithm is chosen by taking a decision obtained as a result of the work of the swarm optimization algorithm.

**2. Neighborhood structure [26].** The neighborhood structure is a mechanism in which new solutions are made through small modifications of already known solutions. Each adjacent (primordial) (Schmidt [35]) solution is derived from an already known one through motion. Even if there is no better solution in the population of the current  $x_n$  with  $V(x_n)$ , then we move to the best solution  $x$  in  $V(x_n)$  or subunit  $V'(x_n) \subset V(x_n)$ , if  $V(x_n)$  is too large for it to be searched effectively. If the area is symmetrical, i.e.,  $x_n$  belongs to the  $V(x_n)$  of  $x \in V(x_n)$ , there is a danger of slipping when we examine  $V(x_n)$  in the next step.

There is a possibility  $x$  to be the best solution in  $V(x_n)$ , which will lead to oscillation between  $x$  and  $x_n$ .

To prevent this situation, we can list in a list  $(x_{n-1}, \dots, x_{n-L})$  called the tabu list, some  $L$  of the last-mentioned solutions. If the solution  $x_n$  is contained in the tabu list, the movement  $x_n \rightarrow x$  becomes a tabu.

Neighborhood Finders are five.

*Neighbor 1.* Randomly select two elements  $e_1$  and  $e_2$  corresponding to different operations in the position vector, and then exchange  $e_1$  and  $e_2$ .

*Neighbor 2.* Randomly select two elements  $e_1$  and  $e_2$  corresponding to different operations in the position vector, and then insert  $e_2$  before  $e_1$ .

*Neighbor 3.* Randomly select two elements  $e_1$  and  $e_2$  corresponding to different operations in the position vector, and then invert the one contained in the positions between  $e_1$  and  $e_2$ .

*Neighbor 4.* Randomly select an item in the second section of the candidate position vector that corresponds to an operation with more than one alternate machine. A machine is selected from alternative machines to replace the current one.

*Neighbor 5.* Randomly, select an item in the second section of the candidate position vector that matches an operation with more than one alternate machine. The current machine for the selected operation will be replaced by the one with the shortest processing time from alternative machines.

**3. Movement.** As a new point of the neighborhood, the best neighbor is chosen which is not a taboo or satisfies any aspirational criteria that makes it not a taboo. "Best," in this case, is the neighbor who has the smallest makespan. If a neighbor is a taboo or no one meets the aspiration criteria, a new neighborhood point is the oldest tabbed neighbor in the list.

Productivity calculation and critical operations can be determined using a simple CPM markup procedure. Such a best estimate makes the special structure of the task for planning of the problems particularly effective [1].

The routing described above leads to a critical mission and it moves to a possible position on an alternate machine. Some obvious limitations, for example, are that the relocated operation cannot begin before its predecessor finishes work. Overall, however, the choice of possible alternative machine positions is complicated in terms of technological constraints.

Experience confirms that it is important to allow zero value moves that go through "flat" areas of the decision space.

In more or in general, it would be desirable to link the dynamic preference indices to any of their displacement types in order to give an opportunity to prioritize the adaptive type of movement when changing the demand.

Routing conditions are allowed for flexible problems at the time of problem introduction and they remain during the search. The search procedure also has zero-value control and moves based on a type of motion; however, as noted earlier, zero values are allowed at any time in this study.

To select the best stroke, the move selection procedure handles each active move. Enter the order of the switch table by invoking the type of motion assessment function. The type of traffic reports a lack of candidates; a type of movement that accounts for optimality leads to the cessation of the demand.

Otherwise, the displacement module reports the best move-type move, either a new allowable move or the oldest taboo move. The comparison of the shift selection is “strictly less than”. The first type of displacement results in an improvement (that is, a movement value) that is usually selected. However, in choosing the best overall move, instead of this, a preference is given to a type of move that counts a new move, even if it is less improved than the existing old-taboo move. The mere comparison of two new movements, or two of the oldest taboo movements, is a preference, based on strict improvement.

**4. Taboo list.** The meaning of the taboo list is to prevent clogging when running the program. Items added to the taboo list are attributes. The taboo list changes after every movement until strategic forgetting occurs.

**5. Aspiration criterion.** The aim of the aspirational criterion is to change the taboo status of a neighbor when it is needed. If the movement gets a better solution than the ones previously received, it is done even if it is a taboo.

**6. Memory structure (adaptive memory).** TS begins in the same way as a local or local search procedure [43], moving from one solution to another until a certain termination criterion is met. Each solution  $x$  has an associated neighborhood  $N(x)$  with  $X$ , and each solution  $x$  in  $N(x)$  is obtained by an operation called movement.

Every simple dwindling method only allows movement to neighborhood solutions that improve the value of the target function, and ends when better solutions cannot be found. The ultimate  $x$  obtained by the declining method is called the local optimum, as long as it is as good as any solution in its surroundings or better than it is. The obvious drawback of the declining method is that such a local optimum is in most cases not a global optimum and it does not minimize  $f(x)$  for all  $x$ .

Unlike a simple decreasing method, where the goal is to reduce  $f(x)$ , TS does something else. TS allows movements that degrade the current value of the target function but the movements are selected from the modified  $N^*(x)$  range. Short-term, medium and long-term memory structures are responsible for the special composition of  $N^*(x)$ . In other words, the modified neighborhood is the result of maintaining a selective history of the solutions found during the search. In TS strategies supporting short-term memory  $N^*(x)$  is a subset of  $N(x)$  and the taboo classification serves to identify elements of  $N(x)$  that are not from  $N^*(x)$ . In TS strategies supporting long-term memory  $N^*(x)$  there may be an extension that contains solutions not necessarily contained in  $N(x)$  as solutions found and evaluated in the past search or identified as

high-quality neighbors of these past solutions. Characterized in this way, TS can be considered as a dynamic surrounding method. This means that the vicinity of  $x$  is not a static set, but a multiple one that can change according to the history of search.

The structure of the area in TS differs from that in the local search for the use of the types of movements that are used in constructive and destructive processes (where the foundations of such movements are referred to as constructive environments and destructive environments). Such extended use of the concept of surroundings strengthens the fundamental perspective of TS, which is to define surroundings in a dynamic manner, which may involve serial or concurrent considerations of many types of movements.

TS uses attribute memory to calculate  $N^*(x)$ . Instead of remembering all decisions, attributes are based on the preservation of attributes. This type of memory remembers information about the decision properties (attributes) that change from one decision to another. The most common attributes are based on recent memory and frequency memory. Recentness, as its name itself, is based on attributes that have changed over the recent past. Frequency typically consists of a proportion of how many times an attribute has changed or not (depending on whether we have a transition or a stay of honor).

Typically, TS based strictly on short-term strategies may allow decision  $x$  to be visited more than once, but as if the corresponding reduced  $N^*(x)$  neighborhood would then be different at each time. With the introduction of long-term considerations, the credibility of repeating a previous neighborhood to obtain an old solution and more generally to visit only a small part of the set  $X$ , is all but not existing.

Recent memory is the most used memory structure in TS. As its name shows, it deals with the attributes of the solution that have changed in the recent past. In order to develop such memory, the selected attributes that appear in recently visited solutions are marked as taboo-active, and decisions that have taboo-active elements or combinations of such elements are those that become taboo. This prevents the decisions of the recent past from belonging to  $N^*(x)$  and the possibility of being re-visited. Other solutions that also offer such taboo-active attributes, likewise, cannot be re-visited. It should be noted that while taboo classification strictly refers to decisions that are forbidden to visit, by virtue of having such taboo-active attributes, movements that lead to such decisions are often also considered taboo.

Frequency memory offers a type of information that adds to the recent memory, expanding the foundation of the selected recommended moves. Both the recent memory and the frequency memory are often divided into subclasses. Also, the frequency can be integrated with the recentness to offer a combined structure that offers penalties and motives for modifying motion assessment.

Frequencies typically consist of proportions whose numerator represents sums expressed by two different measures: changing measure – the number of iterations in which the attribute changes (enter or leaves) the solutions visited on a given trajectory; and resident measure – the number of iterations in which the attribute belongs to the solutions visited on a given trajectory, or the number of specimens in which the attribute belongs to the decisions of a given subset.

Denominators usually represent one of three types of variables: (1) The total number of occurrences of all numbers represented by the numerators (such as the total number of associated iterations), (2) the sum of the numerators, and (3) the largest numerator value. In cases where numerators present weighted sums, some of which potentially negative, denominator (3) is expressed as an absolute value, and denominator (2) is expressed as the sum of absolute values (possibly using a small displacement to prevent the zero denominator). Proportions produce variable frequencies, which watch how often attributes change, and resonant frequencies, which watch how many times attributes become members of the generated solutions. In addition to the reference to such frequencies are the numerator thresholds, which may be useful in indicating when diversification phases are appropriate.

**7. End criterion.** The end criteria can be many and varied. These include, for example, the number of iterations reached, finding an unacceptable solution, reaching the maximum of unapproved movements, and so on.

It is follows Intensification and Diversification Scheme.

The use of recent and frequency memory in TS performs the pre-cyclical prevention function in the search process. More generally, the variations of these two types of memory are designed to give more resilience and strength to the demand.

A key element of TS's adaptive memory is to create a balance between the intensification and the demand diversification. Intensification strategies are based on modifying rules to get more traffic combinations and solutions that are considered to be historically good. They can also trigger a return to attractive regions to be searched more fundamentally. Diversification strategies, on the other hand, look for new attributes and new attribute combinations that were not included in previous solutions, i.e., these strategies start searching in areas where it was not started. It is important to remember that intensification and diversification are not the opposite but rather reinforce one another.

Most types of intensification strategies require tools to identify elite solutions as the basis for introducing good attributes into newly created solutions. Membership in the elite multitude is often determined by setting a threshold that is related to the value of the target function of the best solution found during the search. Two common options for selecting elite solutions have proven to be quite successful. One introduces a diversification measure to ensure that recorded solutions differ to a certain extent from one another and then it erases all short memory before summarizing the best of the saved solutions. The other keeps a linked list that adds a new solution at the end only if it is better than everyone previously seen and keeps the short-term memory that accompanies it.

Diversification is created automatically in TS (to some extent) of short-term memory functions, but is particularly enhanced by ordinary forms of long-term memory. TS diversification strategies are often based on modified selected rules to bring the attributes into a solution that is rarely used. Alternatively, they can introduce such attributes by periodically applying methods that collect subsets of such attributes in candidate solutions to continue the search, or to partially or fully restart the search process. Diverse strategies can be particularly useful when better solutions can only be reached by crossing barriers in the decision space.

The merging of modified selected rules can be guided by the following criminal function:

$$(10) \quad \text{MovementStatus} = \text{MovementStudy} + d * \text{Penalty}.$$

This type of criminal function is fundamental to TS where punishment is often a function of frequency measures and  $d$  is an adjustable diversification parameter. Larger  $d$  values respond to the desire for greater diversity.

#### **Sample TS Algorithm**

##### **Step 1. Initialize:**

- applying a Cycle = 1;
- Placing the TabuCycle = 0.

##### **Step 2.**

- Generate an initial solution;
- Put the makespan and the sequence of jobs on each machine as tabu.

##### **Step 3. Initialize:**

- If Cycle > Maxcycle go to Step 8, otherwise go to Step 4.

##### **Step 4. Neighbor:**

- Find the critical path for the order of job;
- Finding the neighbors of the above critical path;
- Finding the makespan of all neighbors and enter them in the list of neighbors;
- Sort the makespan with neighbors in the neighboring list in ascending order;
- Put in as the best a neighbor who has a minimal makespan;
- Insert the job sequence for each machine as a current job order.

##### **Step 5. Movement:**

- Check that the best neighbor is in the taboo list. If it is not, go to Step 6 or go to check the next neighbor;
- Check that the next neighbor is in the neighbor list. If so, put a next neighbor in the list as the best and go to check for the best neighbor or find the neighbors with the smallest cycle;
- Find neighbors with the minimum cycle in the visit list and delete that neighbor in the visit list;
- Put the top neighbor as the best;
- Re-enter the job order in the visit list as a current job order and go check for best neighbor.

##### **Step 6.**

- Put the best neighbor, his makespan and his job order in the tabu list;
- Put the neighbors with their makespan and their job order in the tabu list.

##### **Step 7.**

- Cycle = Cycle + 1; and go to Step 3.

##### **Step 8.**

- Find the minimal of the makespan in the tabu list;
- Put this makespan and his job order as the best.

##### **Step 9. Exit:**

- The best makespan and the best job order.

## 4. Hybrid Algorithm

PSO is introduced as an optimization technique in the real space  $R$ . The space  $D$  of the FJSSP is discreet. This necessitates a redefinition of PSO for a discreet binary space. The present work uses the presentation by Kennedy and Eberhart [20].

The main idea of the proposed algorithm is as follows: first good solution to be found with the help of heuristics PSO. Then, the “critical path” (the sequence of machine operations completed by the last one) is assumed to become a “tabu”. Then use the TS procedure, trying to improve the gbest generated by the PSO. The phases 3 (PSO), and 4 (TS) are repeated iteratively until the stop position is fulfilled as it is shown in Fig. 1.

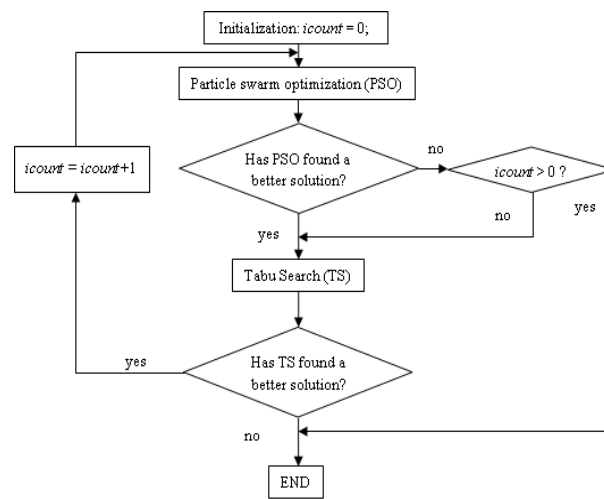


Fig. 1. Flowchart of the algorithm PSO & TS

**Framework of a Hybrid Algorithm.** In this paper, a hybrid optimization algorithm is used through swarm particles and a tabu search to solve FJSSP. The basic framework of a hybrid algorithm is as follows.

**Step 1.** Initialization.

**Step 1.1.** Set the parameters of the hybrid algorithm.

**Step 1.2.** Produce a swarm of initial solutions using the initialization rules given in Section 2.

**Step 2.** Apply the PSO algorithm.

**Step 2.1.** Find the best global and best local particle in the swarm.

**Step 2.2.** Contemplate the global best particle gbest and the locall best pbest for each particle.

**Step 3.** Apply the TS algorithm.

**Step 3.1.** Find the global best particle gbest in the current swarm.

**Step 3.2.** Randomly, select an operation, select another machine for the operation, and update the current global best particle gbest.

**Step 3.3.** Randomly exchange two operations in the current global best particle gbest, and update the current best global particle.

**Step 4.** Apply the fitness function.

**Step 4.1.** Apply a fitness function to the global best particle gbest.

**Step 4.2.** If the stop criterion is met, show the global best particle gbest; otherwise, go to Step 2 to start a new cycle.

## 5. Test results

A set of 12 benchmark test examples are used in this study. The first one is taken from paper by Geiger [12], the second and the eighth – from source Zhang et al. [42], the third – from Low, Yip and Wu [24], the fourth – from Huang [17], the fifth – from Xia and Wu [41], the sixth – from Mesghouni, Hammadi and Borne [28], the seventh – from Udaiyakumar and Chandrasekaran [39], the ninth – from Kasem, Hammadi and Borne [19], the tenth and the twelfth – from Tang et al. [38], and the eleventh – from Li et al. [23]. The optimal solutions of all used test examples are obtained by a genetic algorithm developed by Pesaru [31], created in MATLAB environment (by means of R2012b MATLAB version). Most of the optimal solutions are also published in the corresponding cited references sources. The obtained test results of proposed hybrid PSO&TS algorithm are presented in Table 1, as follows:

Table 1. Results obtained on the test examples

No	Test example	Reference source	Optimal Makespan	Makespan by means of Hybrid PSO & TS	Runs	Times Achieved Makespan	Absolute Error	Makespan by means of Genetic algorithm
1	M2J2O4	Geiger 2012	66	66	30	30	0%	66
2	M3J2O5X	Zhang 2009	53	53	30	30	0%	53
3	M3J2O5L	Low 2006	11	11	30	30	0%	11
4	M3J3O8	Huang 2005	11	11	30	28	0%	11
5	M3J3O9	Xia 2005	45	45	30	4	0%	45
6	M3J4O13	Mesghoini 2004	28	28	30	3	0%	28
7	M4J3O8	Udaiyakumar 2004	12	12	30	14	0%	12
8	M5J2O4	Zhang 2009	5	5	30	30	0%	5
9	M5J3O7	Kasem 2002	11	11	30	30	0%	11
10	M5J3O8	Tang 2011	5	5	30	1	0%	5
11	M5J4O12	Li 2010	11	11	30	1	0%	11
12	M11J3O30	Tang 2011	568	571	30	0	0.52816%	568
Mean error							0.044%	0%

Using the hybrid PSO-TS algorithm the optimal solution is obtained in 11 cases. Only the obtained result for test instance 12 has an error, which is 0.53%. For comparison a genetic algorithm developed by Pesaru [31] has been run. Analysis of the GA performance for optimization of scheduling problems has been done by Mattfeld [27]. Bierwirth and Mattfeld [4] also use of GAs for optimization of dynamic scheduling and rescheduling problems. In 10 of 10 runs the



genetic algorithm generates the optimal solution. The hybrid PSO&TS algorithm has a small deviation from optimality on the example with 30 operations. Nevertheless the obtained results are encouraging. It should be mentioned that the genetic algorithms must be run several times due to their stochastic nature, in order to obtain a minimal deviation from optimality.

For comparison the PSO&TS algorithm is run on every problem for thirty times. In five problem the makespan is reached for thirty times, in M3J3O8 – for twenty eight times, in M4J3O8 – for 14 times, in M3J3O9 and M3J4O13 the makespan is achieved for 4 and, respectively, 3 times. For two problems (M5J3O8 and M5J4O12) the optimal solution has been received for one time and for M11J3O30 – for zero times.

It must be mentioned that with growing of the dimension of the problems decrease the times the solution is reached.

For all cases, the solution is achieved for 15 seconds for one run, only for M11J3O30 the approximate solution – for 30 seconds/run.

## 6. Conclusion

A large variety of different real life problems in practice are formulated as FJSSP optimization problems. In the current paper a hybrid PSO&TS algorithm, based on two popular heuristics, is used to solve the flexible JSSP. The first of them is Particle Swarm Optimization and the second is Tabu Search.

Twelve benchmark examples from the references are used as a test sample. The results show that the mean error is 0.044%, which is encouraging. The time for becoming the results is very small – a few seconds. The algorithm is coded on Matlab.

The developed algorithm will be further tested on instances with larger dimension and it could be refined, i.e. its performance could be improved.

## References

1. Adams, J., E. Balas, D. Zawack. The Shifting Bottleneck Procedure for Job Shop Scheduling. – Management Science, Vol. **34**, 1988, No 3, pp. 391-401.
2. Aïex, R. M., S. Binato, M. G. C. Resende. Parallel GRASP with Path-Relinking for Job Shop Scheduling. – Parallel Computing, Vol. **29**, 2003, pp. 393-430.
3. Balas, E., A. Vazacopoulos. Guided Local Search with Shifting Bottleneck for Job Shop Scheduling. – Management Science, Vol. **44**, 1998, No 2, pp. 262-275.
4. Bierwirth, C., D. C. Mattfeld. Production Scheduling and Rescheduling with Genetic Algorithms. – Evolutionary Computation, Vol. **7**, 1999, No 1, pp. 1-17.
5. Bosejko, W., M. Uchroński, M. Wodecki. Flexible Job Shop Scheduling Problem – Parallel Tabu Search Algorithm for Multi-GPU. – Archives of Control Sciences, Vol. **22**, 2012, No 4, pp. 389-397.
6. Brinkötter, W., P. Brucker. Solving Open Benchmark Problems for the Job Shop Problem. – J. Scheduling, Vol. **4**. 2001, pp. 53-64.
7. Brucker, P. The Job-Shop Problem: Old and New Challenges. – In: Proc. of 3rd Multidisciplinary International Scheduling Conference, Paris, 28-31 August, 2007.  
<http://www.mistaconference.org/2007/papers/The%20Job%20Shop%20Problem%20Old%20and%20New%20Challenges.pdf>
8. Brucker, P., R. Schlie. Job Shop with Multi-Purpose Machine. – Computing, Vol. **45**, 1990, pp. 369-375.

9. Fattahi, P., M. Saidi Mehrabad, F. Jolai. Mathematical Modeling and Heuristic Approaches to Flexible Job Shop Scheduling Problems. – *Journal of Intelligent Manufacturing*, Vol. **18**, 2007, pp. 331-342.
10. Garey, M., D. Johnson, R. Sethi. The Complexity of Flowshop and Jobshop Scheduling. – *Mathematics of Operations Research*, Vol. **1**, 1976, No 2, pp. 117-129.
11. Ge, H., W. Du, F. Qian. A Hybrid Algorithm Based on Particle Swarm Optimization and Simulated Annealing for Job Shop Scheduling. 2007.  
<http://www.paper.edu.cn>
12. Geiger, M. J. Research Report. RR-12-01-01, January 2012, ISSN: 2192-0826, Helmut Schmidt University, Hamburg, Germany.  
<https://d-nb.info/1023241773/34>
13. Glover, F. Tabu Search – Part 1. – *ORSA Journal on Computing*, Vol. **1**, 1989, No 3, pp. 190-206.
14. Glover, F. Tabu Search – Part 2. – *ORSA Journal on Computing*, Vol. **2**, 1990, No 1, pp. 4-32.
15. Glover, F., M. Laguna, R. Marti. *Principles of Tabu Search*, 2003.  
<https://www.uv.es/~rmarti/paper/docs/ts1.pdf>
16. Gulia shki, V., L. Kirilov. A Reference Point Genetic Algorithm for Multi-Criteria Job Shop Scheduling Problems. – In: *Proc. of International Conference on Information Technologies (InfoTech 2015)*, 29-th Issue, R. Romanski, Ed. 17-18 September 2015, Varna, St. St. Constantine and Elena Resort, Bulgaria, pp. 10-18. ISSN: 1314-1023.
17. Huang, Z. A Modified Shifting Bottleneck Procedure for Job Shop Scheduling, 2005.  
<http://www.paper.edu.cn>
18. Gautam, J. V., H. B. Prajapati, V. K. Dabhi, S. Chaudhary. – Empirical Study of Job Scheduling Algorithms in Hadoop MapReduce. – *Cybernetics and Information Technologies*, Vol. **17**, 2017, No 1, Sofia. ISSN: 1311-9702.
19. Kacem, I., S. Hammadi, P. Borne. Pareto-Optimality Approach for Flexible Job-Shop Scheduling Problems: Hybridization of Evolutionary Algorithms and Fuzzy Logic. – *Mathematics and Computers in Simulation*, Vol. **60**, 2002, No 3-5, pp. 245-276.
20. Kennedy, J., R. C. Eberhart. A Diskrete Binary Version of the Particle Swarm Algorithm. – In: *Proc. of Conference Systems Man Cybernetics*, NJ, Piscataway, 1997, pp. 4104-4108.
21. Lawler, E., J. Lenstra, A. Rinnooy Kan, D. Shmoys. Sequencing and Scheduling: Algorithms and Complexity. – In: A. H. G. Rinnooy Kan, S. C. Graves, P. H. Zipkin, Eds. *Logistics of Production and Inventory*, Vol. **4**, Elsevier, 1993, Chapter 9, pp. 445-522.
22. Lenstra, J. K., A. R. Kan, P. Brucker. Complexity of Machine Scheduling Problems. – *Annals of Discrete Mathematics*, Vol. **1**, 1977, pp. 343-362.
23. Li, J., Q. Pan, S. Xie, J. Liang. A Hybrid Pareto-Based Tabu Search for Multi-Objective Flexible Job Shop Scheduling Problem with E/T Penalty. – *Advances in Swarm Intelligence, Lecture Notes in Computer Science*, Vol. **6145**, 2010, pp. 620-627.
24. Low, C., Y. Yip, T.-H. Wu. Modelling and Heuristics of FMS Scheduling with Multiple Objectives. – *Computers & Operations Research*, Vol. **33**, 2006, pp. 674-694.
25. Olteanu, M., N. Paraschiv, P. Koprinkova-Hristova. Genetic Algorithms vs. Knowledge-Based Control of PHB Production. – *Cybernetics and Information Technologies*, Vol. **19**, 2019, No 2. ISSN: 1311-9702.
26. Mastrolilli, M., L. M. Gambardella. Effective Neighborhood Functions for the Flexible Job Shop Scheduling Problem.
27. Mattfeld, D. *Evolutionary Search and the Job Shop: Investigation on Genetic Algorithms for Production Scheduling*. – Physica, Springer, Heidelberg, Germany, 1996.
28. Mesghouni, K., S. Hammadi, P. Borne. Evolutionary Algorithms for Job-Shop Scheduling. – *Int. J. Appl. Math. Comput. Sci.*, Vol. **14**, 2004, No 1, pp. 91-103.
29. Nakanthkumar, R. S., et al. Optimization of Job Shop Scheduling Problem Using Tabu Search Optimization Technique. – *International Journal of Innovative Research in Science, Engineering and Technology*, Vol. **3**, March 2014, Special Issue 3.
30. Nawaz Ripon, K. S., C.-H. Tsang, S. Kwong. An Evolutionary Approach for Solving the Multi-Objective Job-Shop Scheduling Problem. – *Studies in Computational Intelligence (SCI)*, Vol. **49**, 2007, Berlin, Heidelberg, Springer-Verlag, pp. 165-195.

31. P e s a r u, V. Genetic Algorithm-Jobshop Scheduling. 2017.  
<https://de.mathworks.com/matlabcentral/fileexchange/62567-genetic-algorithm-jobshop-scheduling>
32. R a h i m i-V a h e d, R., S. M. M i r g h o r b a n i. A Multi-Objective Particle Swarm for a Flow Shop Scheduling Problem. – Journal of Combinatorial Optimization, Vol. **13**, 2007, No 1, pp. 79-102.
33. R o m a s e v y c h, Y., V. L o v e i k i n. A Novel Multi-Epoch Particle Swarm Optimization Technique. – Cybernetics and Information Technologies, Vol. **18**, 2018, No 3.
34. B o r d b a r, S., P. S h a m s i n e j a d. A New Opinion Mining Method based on Fuzzy Classifier and Particle Swarm Optimization (PSO) Algorithm. – Cybernetics and Information Technologies, Vol. **18**, 2018, No 2.
35. S c h m i d t, K. Using Tabu Search to Solve Job Shop Scheduling Problem with Sequence Dependent Setup Times. 2001.
36. S h a, D. Y., H.-H. L i n. A Multi-Objective PSO for Job-Shop Scheduling Problems. – Expert Systems with Applications, Vol. **37**, March 2010, Issue 2, pp. 1065-1070.  
<http://dl.acm.org/citation.cfm?id=1645843>
37. S h u i b, A., S. S. A. G r a n. Multi-Objectives Optimization Model for Flexible Job Shop Scheduling Problem (FJSSP) with Machines' Workload Balancing. – In: AIP Conference Proceedings, 1974, 020106 (2018).  
<https://doi.org/10.1063/1.5041637>
38. T a n g, J., G. Z h a n g, B. L i n, B. Z h a n g. A Hybrid Algorithm for Flexible Job-Shop Scheduling Problem. – Procedia Engineering, Vol. **15**, 2011, pp. 3678-3683.
39. U d a i y a k u m a r, K. C., M. C h a n d r a s e k a r a n. Application of Firefly Algorithm in Job Shop Scheduling Problem for Minimization of Makespan. – Procedia Engineering, Vol. **97**, 2004, pp. 1798-1807.
40. V a e s s e n s, R. J. M., E. H. L. A a r t s, J. K. L e n s t r a. Job Shop Scheduling by Local Search. – INFORMS J. Computing, Vol. **8**, 1996, pp. 302-317.
41. X i a, W. J., Z. M. W u. An Effective Hybrid Optimization Approach for Multi-Objective Flexible Job-Shop Scheduling Problems. – Computers and Industrial Engineering, Vol. **48**, 2005, No 2, pp. 409-425.
42. Z h a n g, G. H., X. Y. S h a o, P. G. L i, L. G a o. An Effective Hybrid Particle Swarm Optimization Algorithm for Multi-Objective Flexible Job-Shop Scheduling Problem. – Computers and Industrial Engineering, Vol. **56**, 2009, No 4, pp. 1309-1318.
43. Z h a n g, G., at al. An Effective Particle Swarm Optimization for Flexible Job Shop Scheduling Problem. – The Open Automation and Control Systems Journal, Vol. **6**, 2014, pp. 1604-1611.
44. Z h a n g, J., G. D i n g, Y. Z o u, S. Q i n, J. F u. Review of Job Shop Scheduling Research and Its New Perspectives under Industry 4.0. – Journal of Intelligent Manufacturing, Vol. **30**, 2019, Issue 4, No 19, 1809-1830.

*Received: 07.08.2019; Second Version: 03.10.2019; Accepted: 01.11.2019*