A Swarm Intelligence Method Applied to Manufacturing Scheduling

Davide Anghinolfi, Antonio Boccalatte, Alberto Grosso, Massimo Paolucci, Andrea Passadore, Christian Vecchiola, *DIST – Department of Communications Computer and System Sciences, University of Genova*

Abstract—In this paper we present a multi-agent search technique to face the NP-hard single machine total weighted tardiness scheduling problem in presence of sequence-dependent setup times. The search technique is called Discrete Particle Swarm Optimization (DPSO): differently from previous approaches the proposed DPSO uses a discrete model both for particle position and velocity and a coherent sequence metric. We tested the proposed DPSO over a benchmark available online. The results obtained show the competitiveness of our DPSO, which is able to outperform the best known results for the benchmark, and the effectiveness of the DPSO swarm intelligence mechanisms.

***Index Terms*—Particle Swarm Optimization, Swarm Intelligence, Scheduling**

I. Introduction

In this paper we propose a new DPSO approach to face the single machine total weighted tardiness scheduling with sequence-dependent setup times (STWTSDS) problem. Scheduling with performance criteria involving due dates, such as (weighted) total tardiness or total earliness and tardiness (E-T), and that takes into account sequence-dependent setups, is a reference problem in many real industrial contexts. Meeting due dates is in fact recognized as the most important objective in surveys on manufacturing practise, e.g., in [1]. The objective of minimizing the total weighted tardiness has been the subject of a very large amount of literature on scheduling even if sequence-dependent setups have not been so frequently considered. Setups usually correspond to preparing the production resources (e.g., the machines) for the execution of the next job, and when the duration of such operations depends on the type of last completed job, the setups are called sequence-dependent. The presence of sequence-dependent setups greatly increases the problem difficulty, since it prevents the application of dominance conditions used for simpler tardiness problems [2]. The choice of the STWTSDS problem as reference application for the proposed DPSO approach has then two main motivations: first the fact that the solution of single machine problems is often required even in more complex environments [3], and second the absence, to the best authors' knowledge, of any other DPSO approach in literature for the STWTSDS problem. Regarding the latter point, note that the approach in [4] seems to be the only previous DPSO application to the single machine total weighted tardiness (STWT) problem.

The rest of the paper is organized as follows. Section 2 introduces a formal problem definition and provides a general review of the relevant literature for it. Section 3 illustrates the basic aspects of the PSO algorithm, analysing in particular the DPSO approaches previously proposed in the literature. Section 4 then describes the proposed DPSO approach, discussing how it can be applied to the STWTSDS problem and highlighting the new features introduced. Section 5 presents the experimental campaign performed, which is mainly based on the benchmark set generated by Cicirello in [5] and available on the web. Finally, Section 6 draws some conclusions.

II. The STWTSDS Problem

The STWTSDS problem corresponds to the scheduling of n independent jobs on a single machine. All the jobs are released simultaneously, i.e., they are ready at time zero, the machine is continuously available and it can process only one job at a time. For each job j=1,..., n, the following quantities are given: a processing time p_j, a due date d_j and a weight w_j. A sequence-dependent setup time s_{ij} must be waited before starting the processing of job j if it is immediately sequenced after job i. The tardiness of a job j is defined as $T_j=max(0, C_j-d_j)$, being C_j the job j completion time. The scheduling objective is the minimization of the total weighted tardiness expressed as $\sum_{j=1}^{n} w_j T_j$. This problem, denoted as $1/s_{ij}/\sum w_j T_j$, is strongly NP-hard since it is a special case of the $1//\sum w_j T_j$ that has been proven to be strongly NP-hard in [6]. In the literature both exact algorithms and heuristic algorithms have been proposed for the STWTSDS problem or for a slightly different version disregarding the job weights. However, since only instances of small dimensions can be solved by exact approaches, recent research efforts have been focused on the design of heuristics. The apparent tardiness cost with setups (ATCS) heuristic [7] is currently the best *constructive* approach for the STWTSDS problem. Constructive heuristics require a small computational effort, but they are generally outperformed by *improvement*

approaches, based on local search algorithms, and *metaheuristics*, which on the other hand are much more computational time demanding. The effectiveness of such approaches has been largely demonstrated: for example, Potts and van Wassenhove [8] show as simple pair-wise interchange methods outperform dispatching rules for the STWT problem, as well as more recently constructive heuristics appear dominated by a memetic algorithm in [9] or by a hybrid metaheuristic in [10] where a similar parallel machine case is considered. The effectiveness of stochastic search procedures for the STWTSDS is shown in [11], where the authors compare a value-biased stochastic sampling (VBSS), a VBSS with hill-climbing (VBSS-HC) and a simulated annealing (SA), to limited discrepancy search (LDS) and heuristic-biased stochastic sampling (HBSS) on a 120 benchmark problem instances for the STWTSDS problem defined by Cicirello [5]. The literature about applications of metaheuristics to scheduling is quite extended. In [12] an ant colony optimization (ACO) algorithm for the STWTSDS is proposed, which is able to improve about 86% of the best known results for the Cicirello's benchmark previously found by stochastic search procedures in [11]. Recently the Cicirello's best known results have been further independently improved in [13] by means of a GA approach, in [14] with three SA, GA and tabu search (TS) algorithms, and in [15] using an ACO approach; in particular, the new set of best known results established by Lin and Ying [14], which improved more than 71% of the previous best known solutions, was lastly updated by the ACO by Anghinolfi and Paolucci [15] that was able to improve 72.5% of the Lin and Ying solutions.

III. Overview of the Basic PSO Algorithm and its Discrete Variant

Particle Swarm Optimization (PSO) algorithm is a recent metaheuristic approach motivated by the observation of the social behaviour of composed organisms, such as bird flocking and fish schooling, and it tries to exploit the concept that the knowledge to drive the search for optimum is amplified by social interaction. PSO executes a population-based search procedure in which the exploring agents, called *particles*, adjust their positions during time (the particles *fly*) according not only to their own experience, but also to the experience of other particles: in particular, a particle may modify its position with a velocity that in general includes a component moving the particle towards the best position so far achieved by the particle itself to take into account its personal experience, and a component moving the particle towards the best solution so far achieved by any among a set of neighbouring particles (*local neighbourhood*) or by any of the exploring particles (*global neighbourhood*). PSO is based on the *Swarm Intelligence* (SI) concept [16]. This is a particular agent-based modelling technique which mostly relies on the cooperation among large number of simple agents in order to model an autonomous self-organizing system for solving optimization problems. The agents are able to exchange information in order to share experiences, and the performance of the overall multi-agent system (the swarm) emerges from the collection of the simple agents' interactions and actions. PSO has been originally developed for continuous nonlinear optimization ([17]; [18]). The basic algorithm for a global optimization problem, corresponding to the minimization of a real objective function $f(x)$, uses a population (*swarm*) of m particles. Each particle i of the swarm is associated with a position in the continuous n-dimensional search space, $x_i=(x_{i1},\ldots, x_{in})$ and with the correspondent objective value $f(x_i)$ (*fitness*). For each particle i, the best previous position, i.e. the one where the particle found the lowest objective value (*personal best*), and the last particle position change (*velocity*) are recorded and represented respectively as $p_i=(p_{i1},\ldots, p_{in})$ and $v_i=(v_{i1},\ldots, v_{in})$. The position associated with the current smallest function value is denoted as $g=(g_1,\ldots, g_n)$ (*global best*). Denoting with x_i^k and v_i^k respectively the position and velocity of particle i at iteration k of the PSO algorithm, the following equations are used to iteratively modify the particles' velocities and positions:

$$v_i^{k+1} = w \cdot v_i^k + c_1 r_1 \cdot (p_i - x_i^k) + c_2 r_2 \cdot (g - x_i^k) \quad (1)$$

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

where w is the *inertia* parameter that weights the previous particle's velocity; c_1 and c_2, respectively called *cognitive* and *social* parameter, multiplied by two random numbers r_1 and r_2 uniformly distributed in [0, 1], are used to weight the velocity towards the particle's personal best, $(p_i - x_i^k)$, and the velocity towards the global best solution, $(g - x_i^k)$, found so far by the whole swarm. The new particle position is determined in (2) by adding to the particle's current position the new velocity computed in (1). The PSO parameters that must be fixed are the inertia w, the cognitive and social parameters c_1 and c_2, and finally the dimension of the swarm m.

In recent years several studies applying the PSO approach to discrete combinatorial optimization problems appeared in the literature; however, to the best authors' knowledge, none of them faced the STWTSDS problem. PSO has been applied to combinatorial optimization problems, as travelling salesman problem (TSP) [19], vehicle routing problem [20], and scheduling problems ([4]; [12]; [21]; [22]; [23]; [24]). DPSO approaches differ both for the way they associate a particle position with a discrete solution and for the velocity model used; in particular, we could classify DPSO approaches in the literature according to three kinds of solution-particle mapping, i.e., binary, real-valued and permutation-based, and three kinds of velocity model used, i.e., real-valued, stochastic or based on a list of moves. The first DPSO algorithm proposed in [25] is characterized by a binary solution representation and a stochastic velocity model since it associates the particles with n-dimensional binary variables and the velocity with the probability for each binary dimension to take value one. A variation of this DPSO to face flow shop scheduling problems is defined in [26]. A different model is used in [4] to develop a PSO algorithm for the STWT problem and in [27] for the total flowtime

minimization in permutation flow shop problems: using a technique similar to the *random key representation* [28], real values are associated with the particle dimensions to represent the job place in the scheduling sequence and the *smallest position value* (SPV) rule is exploited to transform the particle positions into job permutations. Permutation-based solution-particle mappings are used in [29] for the n-queens problem together with a stochastic velocity model, representing the probability of swapping items between two permutation places, and a mutation operator, consisting of a random swap executed whenever a particle coincides with the *local* (*global*) *best* one.

The velocity models used in all the DPSO approaches above mentioned are either stochastic or real-valued. To the best authors' knowledge the unique example of velocity model based on a list of moves can be found in the DPSO approach for the TSP in [30]. The reason why this kind of model has not been investigated in the scheduling literature may be explained by the main difficulty of defining new appropriate sum and multiplication operators for equations (1) and (2) to make them work in a discrete solution space. Nevertheless, in the following section we propose a new DPSO approach to single machine scheduling based on both a permutation solution-particle representation and on a list-of-moves velocity model.

IV. The Proposed DPSO Approach

To pursuit our purpose we will need to redefine all the arithmetical operators involved in equations (1) and (2). This redefinition will lead in general to build unfeasible sequences (*pseudo-sequences*) that will be fixed and converted in feasible sequences with a procedure called *sequence completion procedure*.

Let us first introduce some notation. In general, a solution x to the problem of scheduling n independent jobs on a single machine is associated with a sequence $\sigma=([1],...,[n])$. In addition we denote with $\varphi_\sigma:\{1,..., n\}\rightarrow\{1,..., n\}$, the mapping between the places in a sequence σ and the indices of the sequenced jobs; for example, if job j is sequenced in the h-th place of σ we have $j= \varphi_\sigma(h)$. In the proposed DPSO we consider a set of m particles; each particle i is associated with a sequence σ_i, i.e., a schedule x_i, and it has a *fitness* given by the cost value $Z(x_i)$. Thus, the space explored by the *flying* particles is the one of the sequences. In the following we introduce a metric for such a space, called *sequence metric*, that is, a set of operators to compute velocities and to update particles' positions consistently.

A. The particle velocity and the sequence metric operators

Given a pair of particles p and q, we define the distance between them as the difference between the associated sequences (*position difference*), i.e., σ_q-σ_p, which corresponds to a list of moves that we call *pseudo-insertion* (PI) *moves*. We denote a PI move as (j, d), where d is the integer displacement that must be applied to job j to *direct* the particle p toward q. Roughly speaking, assuming for example that $j=\varphi_\sigma(h)$, a positive displacement d corresponds to a towards-right move that extracts job j from its current place h and reinserts it in place $\min(h+d, n)$ in the sequence, so generating a new sequence σ' such that $j=\varphi_{\sigma'}(\min(h+d, n))$, and a corresponding solution x'; analogously, a negative displacement $-d$ corresponds to a towards-left extraction and reinsertion move that generates a new sequence such that $j=\varphi_{\sigma'}(\max(h-d, 0))$. The difference between the positions of two particles p and q defines a *velocity* v, which consequently is a set of PI moves; then, applying the PI moves in v to p we can move this particle to the position of particle q. The following example would simply illustrate this concept. Let the number of jobs n=4 and the sequences corresponding to the positions of two particles p and q respectively σ_p=(1,2,3,4) and σ_q=(2,3,1,4); then, the velocity associated with the difference between the two positions is $v=\sigma_q-\sigma_p=\{(1,2),(2,-1),(3,-1)\}$; here the PI move (1, 2) denotes that job 1 must be delayed (moved towards-right) of 2 places in the sequence to direct particle p towards q. Note that a velocity can include at most a single PI move for a given job. The reason why we denote as "pseudo-insertion" such kind of moves is that, as detailed in the following, in general the rule used to apply the PI moves in a velocity to a sequence may fail to produce a feasible sequence, but it may produce a so-called *pseudo-sequence*, and we need to introduce a final *sequence completion procedure* to correctly implement equation (2) in the sequence metric.

The position-velocity sum operator applies one PI move composing the velocity at a time, first to the initial sequence and then to the *pseudo-sequences* successively obtained, hereafter denoted by π.

B. The sequence completion procedure

In general, the pseudo-sequences produced do not correspond to feasible sequences since some sequence places may be left empty whereas some others may contain a list of jobs. If, for example, we apply the move (1,2) to σ_p=(1,2,3,4), we obtain the pseudo-sequence π=(-,2,[3,1],4), where "- " denotes that no job is assigned to the first place of π, whereas [3,1] represents the ordered set of jobs assigned to the third place of π. Let us denote with $\pi(h)$ the ordered set of items in the h-th place of the pseudo-sequence π; with *pull*(s) the function that extracts the first element from an ordered set s, and with *push*(i, s) the function that inserts the element i at the bottom of the set s. Then, in order to convert a pseudo-sequence into a feasible sequence, the sequence completion procedure manages $\pi(h)$ as a first-in-first-out (FIFO) list, as reported in Fig. 1.

As an example, the behaviour of such a procedure for a pseudo-sequence π=([1,3],-,-,[4,2]) is shown in Fig. 2.

```
Input: π a pseudo-sequence
Output: σ a feasible sequence
for each h=1,...,n
{
   if |π(h)|=1 skip;
   else if |π(h)|=0
   {
      repeat
         k=h+1;
      while k<n and |π(k)|=0
      π(h)=pull(π(k));
   }
   else if |π(h)|>1
   {
      while |π(h)|>1
         push(pull(π(h), π(h+1));
   }
}
σ=π;
```

Fig. 1: The sequence completion procedure.

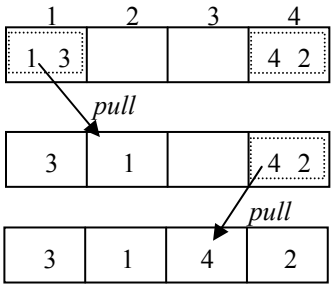

Fig. 2: An example of sequence completion procedure execution.

The procedure considers one place at a time of π starting from the first one on the left; since an ordered set of jobs is encountered in place h=1, then the first job is extracted and reinserted in the first following empty position (h=2), thus, the pseudo-sequence is updated as (3,1,-,[4,2]); then place h=2 is skipped because it contains just one job. In h=3 an empty place is encountered, so the procedure extracts a job from the next not empty place, here place 4 containing the FIFO list [4,2], and reinserts it there; after this step the final feasible sequence (3,1,4,2) is obtained.

It is easy to verify that the iterated application of the extract-reinsert operator in (3) to compute σ_p+v in the case of the first example where σ_p=(1,2,3,4), σ_q=(2,3,1,4), and v={(1,2),(2,-1),(3,-1)} directly gives the target sequence σ_q since π_0=(1,2,3,4), π_1=(-,2,[3,1],4), π_2=(2,-,[3,1],4) and finally π_3=(2,3,1,4).

A velocity v can be summed to another velocity v' producing a new velocity w. This is a different sum operator (*velocity sum*) that generates the resulting velocity as the union of the moves in v and v'. Any job can appear only once in the set of pseudo-moves defining a velocity; therefore, if v and v' include respectively (j, d) and (j, d'), then the resulting sum w must include the pseudo-insertion move (j, $d+d'$). Note that if $d+d'$=0 the move is removed from the list.

Finally, a velocity v can be multiplied by a real positive constant c (*constant-velocity multiplication*) generating a new velocity $w=c\cdot v$. We devised the following constant-velocity multiplication rule according to which the constant c modifies the displacement values of the pseudo-moves included in $v=\{(j_1,d_1),...,(j_s,d_s)\}$; in particular, this rule produces a velocity $w=\{(j_1, c\cdot d_1),...,(j_s, c\cdot d_s)\}$.

C. The overall DPSO algorithm

The very high level structure of the developed DPSO algorithm is given in Fig. 3. In the following we will describe each step in the detail.

```
Initialization;
While <termination condition not met>
{
   For each particle p belonging to P
   {
      Update particle velocity;
      Update particle position;
      Compute particle fitness;
   }
   Intensification phase;
   Update best references;
}
```

Fig. 3: The overall D-PSO algorithm.

Initialization. An initial sequence σ_i^0, i=1,..., m, (i.e., an initial solution x_i^0) is assigned to each of the m particles. In particular, we use three different constructive heuristics, the earliest due date (EDD), the shortest processing time (SPT), and the apparent tardiness cost with setups (ATCS) to generate three different starting sequences. Then, a set of v_i^0, i=1,...,m initial velocities is randomly generated and associated with the particles. Finally, the initial position for each particle i is produced first randomly selecting one among the first three starting sequences and then summing the correspondent initial velocity v_i^0.

Velocity and position update. At iteration k, each particle i first computes the following components: the inertial velocity (*iv*), the directed to personal best velocity (*pv*), and the directed to global best velocity (*gv*), according to the following equations:

$$iv_i^k = w\cdot v_i^{k-1} \tag{3}$$

$$pv_i^k = c_1 r_1 \cdot (p_i - \sigma_i^{k-1}) \tag{4}$$

$$gv_i^k = c_2 r_2 \cdot (g - \sigma_i^{k-1}) \tag{5}$$

where r_1 and r_2 are independent random numbers extracted from U[0,1].

Then the particle velocity at iteration k is updated with a procedure that separately sums to the current particle position

each velocity component one at a time (in the *iv*, *pv*, *gv* order), thus moving the particle through a set of intermediate sequences. For example, denoting with *is* the intermediate sequences, we must execute three sums, $is_1 = \sigma_i^{k-1} + iv_i^k$, $is_2 = is_1 + pv_i^k$ and $\sigma_i^k = is_2 + gv_i^k$ in order to update the position of a particle.

Finally, the schedule x_i^k associated with the updated particle position σ_i^k is determined by a straightforward timetable procedure, and the fitness $Z(x_i^k)$ is computed. Note that if the velocity for a particle becomes null then it is reinitialized by a random restart.

Intensification phase. After all the particles have updated their position and computed their fitness at an iteration *k*, an intensification phase is performed consisting of a local search (LS) exploration that starts from the best solution found by the particles in the current iteration. We adopt a stochastic LS (S-LS) algorithm similar to the one in [4], which in turn is based on a variant of the variable neighbourhood search (VNS) [31].

The S-LS algorithm performs a random neighbourhood exploration allowing an alternation of random insert and swap moves with a maximum number of random restarts bounded by n/5; thus the overall complexity of the LS algorithm is $O(n^3)$. After the intensification phase, the solution obtained by the S-LS algorithm is substituted to the starting $x_{i^*}^k$ for the particle i^*, whose position and fitness are updated accordingly.

Update of the best references. After the completion of the intensification phase, the global and the personal best position for the particles may be updated.

V. Experimental Analysis of the Algorithm

We coded the DPSO algorithm in C++ and implemented it on a Pentium IV, 2.8 GHz, 512 Mb PC. We extensively tested the behaviour of the proposed DPSO through an experimental campaign mainly based on the benchmark due to Cicirello [5], which is available on the web at http://www.cs.drexel.edu/~cicirello/benchmarks.html. This benchmark is made of a set of 120 STWTSDS problem instances with 60 jobs. We compared our DPSO algorithm with the following three sets of best reference solutions for the considered benchmark: (a) a set including the overall aggregated best known results, denoted with OBK, mostly composed by the solutions yielded by the SA, GA and TS algorithms in [14] with the addition of few best solutions from the ant colony optimization (ACO) algorithm in [12], and taking also into account the best solutions from the GA in [13]; (b) the set of best results obtained by the ACO algorithm in [12], denoted with ACO-LJ; (c) the most recent set of the best results produced by the authors with an ACO approach denoted with ACO-AP [15].

During all the experimental campaign, we set the number of particles *m*=120 and we adopted the same termination criterion used in [14] fixing the maximum number of fitness function evaluation = 20,000,000. After a preliminary experiment campaign we also fixed c_1=1.5, c_2=2.0 and *w*=1.0. Please note that these parameters seemed to be not much sensitive, as also other configurations gave results statistically not different. We executed 10 runs for each instance and then we computed the best results, summarized in Table 1. This table reports the average percentage deviations (*Avg dev*), the related 95% confidence (*Conf*), the percentage number of improved (*Impr sol*) and identical (*Ident sol*) solutions found by DPSO with respect to the three sets of reference solutions. Table 1 clearly shows that the best DPSO solutions outperform on the average the OBK and the ACO-LJ ones, while they are substantially equivalent to the ones in ACO-AP approach.

	Avg dev	Conf	Impr sol	Ident sol
OBK	-2.80	1.72	70.83	18.33
ACO-LJ	-4.60	1.91	65.00	13.33
ACO-AP	-0.24	1.42	31.67	26.67

Table 1: The best results of the DPSO with respect to three solution sets (%)

The dominance of DPSO in the first two comparisons, also witnessed by the 95% confidence results, was also confirmed by statistically significance tests.

At the website http://www.discovery.dist.unige.it/DPSO_best.html the complete best results for each instance can be found with every objective function value and sequence of jobs.

A. The evaluation of the importance of the swarm intelligence mechanisms

In order to finally verify the effectiveness of using swarm intelligence mechanisms in exploring the solution space, we developed a modified version of our DPSO, denoted as Random Particle Search (RPS), removing from the algorithm every memory and particle interaction mechanism. The RPS, starting from the set of solutions initially associated with the *m* particles, executes at each iteration a random position update for each particle and an intensification step with the S-LS for the particle position correspondent to the best solution found in the iteration. Differently from the DPSO, the RPS updates the particle positions computing a random velocity as follows: for each particle dimension, i.e., job *j* in the sequence of the associated solution, a pseudo-insertion move (*j*, *d*) is determined by stochastically generating the job displacement *d* from a normal distribution $N(\mu, \sigma^2)$, with mean μ=0 and standard deviation σ fixed as algorithm parameter. The developed RPS can be viewed as a sort of multiple iterated local search method that uses the velocity concept from DPSO in order to perturb the current solutions, but that does not include any "swarm" interaction mechanism as well as PSO memory structures (personal or global best). We tested three RPS configurations characterized by a different value for the parameter σ, fixing $\sigma \in \{4, 6, 18\}$, executing 10 runs for each configuration on the Cicirello's benchmark, then computing for each instance the best average result over the three RPS configurations. Then we compared the RPS results with the average DPSO solutions finding that the RPS produced an

average percentage deviation from the DPSO of 12.15%, with a 95% confidence of 9.45%. From such results RPS appears dominated by the DPSO and this fact clearly confirms the fundamental role of the DPSO swarm intelligence mechanisms.

VI. Conclusions

In this paper we describe a new DPSO algorithm that we used to face the NP-hard STWTSDS problem. To our best knowledge, this should be the first application of a discrete PSO metaheuristic to this class of scheduling problem. Differently from previous approaches in the literature where PSO has been applied to scheduling problems, our DPSO adopts a discrete model both for particles and velocities, respectively corresponding to job sequences and list of so-called pseudo-insertion moves.

The experimental tests performed on the Cicirello's benchmark demonstrate the competitiveness of the proposed DPSO; in particular, we can highlight the ability of the DPSO of generating excellent average results, as well as its very limited dependency from the parameter values, which makes the algorithm tuning not critical. Finally, we showed the effectiveness of this swarm intelligence method, since turning off interaction and memory mechanisms of agents the performance of the algorithm deteriorates significantly.

References

[1] Wisner, J., & Siferd, S. (1995). A Survey of U.S. Manufacturing Practices in Make-to-Order Machine Shops. Production and Inventory Management Journal , 36, 1-7.

[2] Rubin, P., & Ragatz, G. (1995). Scheduling in a sequence dependent setup environment with genetic search. Computers & Operations Research , 22, 85–99.

[3] Pinedo, M. (1995). Scheduling: Theory, Algorithms, and Systems. Englewood Cliffs, NJ: Prentice Hall.

[4] Tasgetiren, M., Sevkli, M., Liang, Y., & Gencyilmaz, G. (2004). Particle swarm optimization algorithm for single machine total weighted tardiness problem. Proceedings of the IEEE congress on evolutionary computation, vol.2, p. 1412–1419. Portland.

[5] Cicirello, V. (2003). Weighted tardiness scheduling with sequence-dependent setups: a benchmark library. Carnegie Mellon University, USA, Technical Report of Intelligent Coordination and Logistics Laboratory, Robotics Institute.

[6] Lawler, E. (1997). A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics , 1, p. 331–342.

[7] Lee, Y., Bhaskaran, K., & Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transaction , 29, 45-52.

[8] Potts, C., & van Wassenhove, L. (1991). Single machine tardiness sequencing heuristics. IIE Transactions , 23, 346–354.

[9] França, P., Mendes, A., & Moscato, P. (2001). A memetic algorithm for the total tardiness single machine scheduling problem. European Journal of Operational Research (132), 224-242.

[10] Anghinolfi, D., & Paolucci, M. (2007). Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. Computers & Operations Research (34), 3471-3490.

[11] Cicirello, V., & Smith, S. (2005). Enhancing stochastic search performance by value-based randomization of heuristics. Journal of Heuristics (11), 5–34.

[12] Liao, C., & Juan, H. (2007). An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Computers & Operations Research (34), 1899-1909.

[13] Cicirello, V. (2006). Non-Wrapping Order Crossover: An Order Preserving Crossover Operator that Respects Absolute Position. Proceeding of GECCO'06 Conference, (p. 1125-1131). Seattle, Washington, USA.

[14] Lin, S., & Ying, K. (2006). Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics. The International Journal of Advanced Manufacturing Technology .

[15] Anghinolfi, D., & Paolucci, M. (2007). A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem. accepted for publication on International Journal of Operations Research .

[16] Kennedy, J., & Eberhart, R. (2001). Swarm Intelligence. San Francisco: Morgan Kaufmann Publishers.

[17] Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. Proceeding of the 1995 IEEE International Conference on Neural Network (p. 1942-1948). IEEE Press.

[18] Abraham, A., Guo, H., & Liu, H. (2006). Swarm Intelligence: Foundations, Perspectives and Applications. Swarm Intelligence in Data Mining, Studies in Computational Intelligence (series) .

[19] Pang, W., Wang, K., Zhou, C., & Dong, L.-J. (2004). Fuzzy discrete particle swarm optimization for solving traveling salesman problem. Proceedings of the 4th International Conference on Computer and Information Technology (p. 796 – 800). IEEE CS Press.

[20] Chen, A., Yang, G., & Wu, Z. (2006). Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem. Journal of Zhejiang Univ. SCIENCE A (7), 607-614.

[21] Lian, Z., Gu, X., & Jiao, B. (2006). A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. Applied Mathematics and Computation (175), 773-785.

[22] Lian, Z., Gu, X., & Jiao, B. (2006). A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. Applied Mathematics and Computation (183), 1008-1017.

[23] Allahverdi, A., & Al-Anzi, F. (2006). A PSO and a Tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. Computers & Operations Research (33), 1056–1080.

[24] Parsopoulos, K., & Vrahatis, M. (2006). Studying the Performance of Unified Particle Swarm Optimization on the Single Machine Total Weighted Tardiness Problem. Lecture Notes in Artificial Intelligence (4304), 760-769.

[25] Kennedy, J., & Eberhart, R. (1997). A discrete binary version of the particle swarm algorithm. Proceedings of the International Conference on Systems, Man, and Cybernetics. vol. 5, p. 4104–4108. IEEE Press.

[26] Liao, C.-J., Tseng, C.-T., & Luarn, P. (2007). A discrete version of particle swarm optimization for flowshop scheduling problems. Computers & Operations Research (34), 3099-3111.

[27] Tasgetiren, M., Liang, Y.-C., Sevkli, M., & Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research (177), 1930-1947.

[28] Bean, J. (1994). Genetic algorithm and random keys for sequencing and optimization. ORSA Journal on Computing (6), 154-160.

[29] Hu, X., Eberhart, R., & Shi, Y. (2003). Swarm intelligence for permutation optimization: a case study of n-queens problem. Proceedings of the 2003 IEEE Conference on Swarm Intelligence Symposium (SIS '03) (p. 243-246). IEEE Press.

[30] Clerc, M. (2004). Discrete Particle Swarm Optimization. Onwubolu GC, Babu BV (Eds), New Optimization Techniques in Engineering , 219-240.

[31] Mladenovic, N., & Hansen, P. (1997). Variable Neighbourhood Search. Computers & Operations Research (24), 1097-1100.