International Journal of Computational Intelligence Systems, Vol. 6, No. 4 (July, 2013), 684-699

A Particle Swarm Optimization Algorithm for Scheduling Against Restrictive Common Due Dates

Andreas C. Nearchou*
Department of Business Administration, University of Patras,
26 500 Patras, Greece

Sotiris L. Omirou
Department of Mechanical Engineering, Frederick University,
Pallouriotissa 1036, Nicosia, Cyprus,
E-mail: eng.os@frederick.ac.cy

Received 1 April 2011
Accepted 14 December 2012

Abstract:

Focusing on the just-in-time (JIT) operations management, earliness as well as, tardiness of jobs' production and delivery should be discouraged. In accordance to this philosophy, scheduling problems involving earliness and tardiness penalties are very critical for the operations manager. In this paper, a new population heuristic based on the particle swarm optimization (PSO) technique is presented to solve the single machine early/tardy scheduling problem against a restrictive common due date. This type of scheduling sets costs depending on whether a job finished before (earliness), or after (tardiness) the specified due date. The objective is to minimize a summation of earliness and tardiness penalty costs, thus pushing the completion time of each job as close as possible to the due date. The problem is known to be *NP*-hard, and therefore large size instances cannot be addressed by traditional mathematical programming techniques. The performance of the proposed PSO heuristic is measured over benchmarks problems with up to 1000 jobs taken from the open literature, and found quite high and promising in respect to the quality of the solutions obtained. Particularly, PSO was found able to improve the 82% of the existing best known solutions of the examined benchmarks test problems.

Key words: Meta-heuristics, swarm intelligence, combinatorial optimization, job scheduling, just-in-time.

*Corresponding author. E-mail address: nearchou@upatras.gr

1. Introduction

Sequencing and scheduling problems involving due dates, play a crucial role in real-world production and operations management. This type of scheduling sets penalty costs depending on whether a job finished before (earliness), or after (tardiness) the specified due date. In the last two decades, much research effort has been spent on the study of earliness and tardiness penalties in scheduling[1,2] due to its accordance with the principles of just-in-time (JIT) operations management. JIT adopts the notion that jobs must be completed as close as possible to their due date, neither too early, nor too late. Early jobs result in inventory holding costs, while late jobs result to penalties such as loss of customer goodwill and loss of orders. Therefore, earliness as well as tardiness of jobs should be discouraged.

This paper deals with the single-machine early/tardy scheduling problem (SMETSP) of a set of jobs with a common due date (CDD) and objective the minimization of the jobs' total earliness and tardiness. SMETSP belongs to a large class of scheduling problems[2] formally classified as n/1//ET. This class consists of problems with distinct due dates, problems with a CDD, problems with single, or multiple performance measures being either linear or non-linear, etc. Some of these problems can be solved in polynomial time using traditional mathematical programming methods, while many other are known to be intractable. One assumption often made about CDD is that it is sufficiently large so that it does not constrain the scheduling of the jobs. This is known as the unrestricted version of the problem (consequently CDD is called unrestrictive). The restricted version of the problem is obviously harder since the value of CDD is small enough to constrain the scheduling process.

Even the simplest formulation of SMETSP leads to an *NP*-hard combinatorial optimization problem (COP)[1,2], and thus it seems fair, large size instances of the problem to be addressed by the means of heuristics. Since the pioneer work of Kanet[3] which deals exclusively with the special case when the earliness and tardiness penalties are equal to one, many approximate algorithms have been proposed for various versions of the basic problem. Hall[4] and Bagchi *et al.*[5,6] proposed algorithms for the absolute deviation SMETSP, which involve minimizing the sum of absolute deviations of the job completion times from a CDD. Hall and Posner[7] examined the unrestricted weighted earliness and tardiness problem. De *et al.*[8], proposed a greedy randomized adaptive heuristic for the unrestricted problem with different penalties to find both the optimal due date and the optimal sequence of the jobs. Hoogeveen and van de Velde[9] addressed the unrestricted SMETSP with 'almost' CDD using a dynamic programming algorithm. More about algorithms on scheduling problems involving due dates can be found in two comprehensive reviews considered by Baker and Scudder[1] and Cheng and Gupta[10] respectively. These reviews cover the results published before 1990. Recent material and results can be found in the survey paper of Gordon *et al.*[11].

The majority of the proposed algorithms for SMETSP addressed instances of the problem with a small number of jobs, up to 25 or 50 jobs. For instance, Abdul-Razaq and Potts[12] solved to optimality problems with up to 25 jobs using a branch and bound algorithm. Souza and Wolsey[13] proposed branch and bound algorithms for solving a class of four different scheduling problems (including SMETSP) with 20 and 30 jobs. Almeida and Centeno[14] addressed SMETSP with up to 50 jobs via a composite algorithm that combines steepest-descent, simulated annealing and tabu-search. Recently, the use of meta-heuristics such as tabu-search (see Refs. 15-17), genetic algorithms[15], simulated annealing[18], differential evolution[19,20], ant colony optimization[21-23], artificial immune systems[24] enable researchers to address effectively large size instances of the problem. Among them, James[17], Feldmann and Biskup[18], Nearchou and Omirou[19], Nearchou[20], and Lee *et al.*[22] addressed the restricted SMETSP with general earliness and tardiness penalties. Moreover, Biskup and Feldman[25] generated a set of benchmarks for SMETSP together with their upper bounds on the optimal objective functions.

The current paper investigates the application of the particle swarm optimizer (PSO) algorithm on the

restricted SMETSP with general earliness and tardiness penalties. PSO is one of the latest meta-heuristics introduced by Eberhart and Kennedy[26] for optimization over continuous spaces and its application to discrete COPs is still limited. Previous PSO-based approaches on single-machine scheduling problems can be found in the recent works of Anghinolfi and Paolucci[27] and Low *et al.*[28]. The former work tackled the classical single-machine weighed tardiness scheduling problem with sequence-dependent setup times; while, the latter considered the single-machine scheduling problem with periodic maintenance. To the best of our knowledge, the only researchers who addressed scheduling problems involving due dates using the PSO algorithm have been Pan *et al.*[29]. Our PSO approach differs from these studies in two main points: first in the developed encoding mechanism; that is, in the way 'particles' are represented and mapped into actual scheduling solutions; and second, in the way of controlling the 'particles' velocities of the entire swarm.

The performance of the proposed PSO algorithm is examined over the most restricted instances against CDDs of the Biskup and Feldmann (2001)'s benchmarks[25]; including 140 instances in total ranging from 10 to 1000 jobs. The results obtained are of high quality since new upper bounds have been introduced by PSO in the 82% of the examined benchmarks instances. The motivation behind the idea of applying PSO on SMETSP goes back to our previous works[19,20] in which we tackled the problem with high success through another meta-heuristic namely, differential evolution. Since the last five years there is a considerably research interest to tackle hard COPs through PSO-based approaches, it was decided to investigate the behavior and the performance of PSO on SMETSP too. Therefore, a major objective of this work is to show (through extended analysis of critical aspects, such as solution space, representation, encoding and decoding mechanisms) how PSO, a meta-heuristic initially proposed as global optimizer over continuous search spaces can be applied with success on discrete COPs too.

The rest of this paper is organized as follows: Section 2 states the problem. Section 3 gives a description of the basic PSO algorithm for optimization over continuous search spaces. Section 4 introduces the way PSO can be applied on SMETSP (a typical discrete COP), while Section 5 presents and discusses the results of the experimental evaluations of the algorithm. Finally, Section 6 summarizes the contribution of the paper and states some directions for future work.

2. Problem formulation

SMETSP can be formally defined as follows: consider n jobs (numbered 1,2,…,n) to be processed without interruption on a single machine that can handle only one job at a time. Each job j (j=1,…,n) is available at time zero, requires a positive processing time p_j and ideally must be completed exactly on a specific (common for all jobs) due date d. Penalties are incurred whenever a job is completed before or after this due date. Therefore, an ideal schedule is one in which all jobs finish on the specific due date. Assuming that C_j is the completion time of job j, then the earliness and tardiness of job j are given by the relations, $E_j = \max(0, d - C_j)$ and $T_j = \max(0, C_j - d)$, respectively, for all j=1,…,n. The objective is therefore to find a processing order of the n jobs that minimizes

$$\sum_{j=1}^{n} \left(a_j E_j + b_j T_j \right) \tag{1}$$

where a_j, b_j (j=1,…,n) are the earliness and tardiness (nonnegative) penalties, respectively, for job j and constitute data input to the scheduling problem.

Penalties in Eq. (1) can be measured in different ways resulting in several variations of the basic SMETSP. A CDD d is called unrestrictive when $d \geq \sum p_j$ (j=1,…,n) holds, otherwise is called restrictive. Moreover, d is also called unrestrictive when it constitutes a decision variable for the problem. Consequently, one can refer to the problem as either unrestricted or restricted SMETSP.

The basic assumptions with SMETSP can be summarized as follows:

- Jobs' processing times are deterministic.
- Machine breakdown and maintenance are neglected. The machine is continuously available and never kept idle while there are jobs waiting to proceed.
- The machine processes only one job at a time.
- No setups between jobs are assumed.
- Jobs are known in advance.

- Each job is available for processing at time zero.
- No job pre-empt is permitted.
- Jobs are independent without precedence or other constraints.
- All jobs must be completed on a particular common due date.

2.1. Properties for the unrestricted CDD SMETSP

There exists an optimal solution to the unrestricted CDD SMETSP having the following properties:

a) There is no inserted idle time in the schedule.
b) The schedule, is V-shaped (see Fig. 1), i.e., early jobs are sequenced in non-increasing order of p_j/a_j ('\-shaped' format), and late jobs are sequenced in non-decreasing order of p_j/b_j (j=1,…,n) ('/-shaped' format).
c) One job is completed exactly on the due date.
d) The q-th job in the sequence completes on the due-date d, where q is the smallest integer satisfying the inequality $\sum_{j=1}^{q}(a_j + b_j) \geq \sum_{j=1}^{n} b_j$

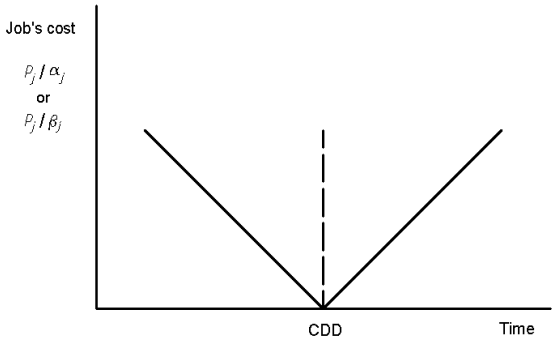


Fig. 1: The V-shaped property

2.2 Properties for the restricted CDD SMETSP

For the restricted SMETSP with general earliness and tardiness penalties there is an optimal schedule with the following properties:

a) No idle times are inserted between consecutive jobs[30].
b) The schedule is V-shaped, but a straddling job may exist, i.e., a job whose execution starts before and finished after the due date[25].
c) The processing time of the first job either starts at time zero, or one job is completed at the due date[25].

Obviously, the restricted SMETSP is much more complex than the unrestricted version since all its variations result to an *NP*-hard COP[11]. There is a gap in the related literature for algorithms taking into account the third property of the restricted SMETSP. That is, the case where the first job in an optimal schedule might not start at time zero; thus, excluding optimal schedules a priori. This characteristic is investigated in Biskup and Feldmann[25]. It is worth pointing out that, there is no such limitation with the proposed PSO algorithm. Particularly, the use of a suitable encoding scheme enables PSO taking into account the case of the third property in the generated SMETSP solutions.

3. The particle swarm optimization (PSO) algorithm

PSO is a stochastic population heuristic introduced by Eberhart and Kennedy[26] for continuous non-linear function optimization. According to its founders, PSO has roots in two recent 'intelligent' optimization methodologies: in artificial life and in evolutionary computation. In regard to artificial life, PSO has ties with bird flocking and fish schooling theories, while in regard to evolutionary computation has similarities with genetic and evolutionary algorithms. Since its invention, PSO has been applied with success on various COPs such as, the unit commitment problem[31], the traveling salesman problem[32], the task assignment problem[33], an optimal operational path finding for automated drilling operations[34], a multi-objective order planning production problem in steel sheets manufacturing[35], scheduling problems involving due-dates[29], the shortest path problem[36], etc. Recently, its application has been extended on scheduling problems such as, flow-shop scheduling problems[37-41], the single-machine total weighting tardiness problem[27,42], the single machine scheduling problem with periodic maintenance[28], the two-stage assembly-scheduling problem[43], and job-shop scheduling problems[44,45].

Assuming the problem of minimizing a real-valued function $f(x)$, $x \in \Omega \subset \Re^D$ (Ω is assumed to be the feasible search space of the problem), PSO utilizes a set (called swarm) of Ns particles as a population to search

Co-published by Atlantis Press and Taylor & Francis

Ω toward the global optimal solution. Each particle represents a potential solution to the optimization problem flying in the *D*-dimensional search space, modifying (iteration by iteration) its position according to its own best position in history and that of its neighbors. Each particle i (i=1,2,…,*Ns*) has three attributes: its current position $x_{i,k}$, its personal best position achieved so far $x_{i,k}^{pbest}$, and its current velocity $v_{i,k}$. Note that each one of these attributes is a *D*-dimensional parameter vector. The index *k* denotes the iteration number of the algorithm. The initial population (k = 0),

$$S = \{x_{1,0}, x_{2,0}, \mathbf{K}, x_{Ns,0}\}, \tag{2}$$

is taken to be uniformly distributed in the search region using the following formula:

$$x_{i,0}^{j} = x_{\min} + random \cdot (x_{\max} - x_{\min}), \tag{3}$$

where, $x_{i,k}^{j}$ is the position value of the i-th (i=1,2,…,*Ns*) particle with respect to the j-th (j=1,2,…,*D*) dimension. $x_{\min}$ and $x_{\max}$ are user defined bounds, and *random* is a uniform random number in (0,1). Similarly, the initial velocities of the particles are generated using the formula

$$v_{i,0}^{j} = v_{\min} + random \cdot (v_{\max} - v_{\min}), \tag{4}$$

with $v_{\min}$, $v_{\max}$ user-defined fixed bounds, and *random* a uniform random number in (0,1).

At each iteration *k*, all particles in *S* are targeted for replacement. This is achieved by performing the following steps:

STEP 1:

For each particle i (i=1,2,…,*Ns*)

1.1) Evaluate its objective function $f(x_{i,k})$.

1.2) Determine its personal best position $x_{i,k}^{pbest}$ as in the following:

if k=0 **then** $x_{i,0}^{pbest} = x_{i,0}$

else if $f(x_{i,k}) < f(x_{i,k}^{pbest})$ **then** $x_{i,k}^{pbest} = x_{i,k}$ (5)

STEP 2:

Determine the global best position x_{k}^{gbest} corresponding to the best objective function value among the population of the particles (i.e., the whole swarm).

STEP 3:

For each particle i (i=1,2,…,*Ns*) update its velocity $v_{i,k}$ as in the following:

$$v_{i,k} = c_1 \cdot r_1 \cdot (x_{i,k}^{pbest} - x_{i,k}) + c_2 \cdot r_2 \cdot (x_{k}^{gbest} - x_{i,k}) + w_k \cdot v_{i,k-1} \tag{6}$$

where, c_1 and c_2 are called *cognitive* and *social* parameters, respectively, and r_1, r_2 are uniform random numbers drawn in (0,1), c_1 and c_2 (also known with the term *learning factors*) represent the attraction that a particle has toward to its own success (c_1), or that of its neighbors (c_2). In other words, c_1 is a weight factor representing the attraction toward $x_{i,k}^{pbest}$, while c_2 the attraction toward x_{k}^{gbest}. Both of them are usually defined to be constants during the execution of the algorithm. w_k in Eq. (6) is the inertia weight factor which gadgets the effect of the old velocity onto the new one. Generally, w_k is updated by the linear equation:

$$w_k = \Theta \times w_{k-1} \tag{7}$$

where, Θ is a decrement user-defined constant factor.

STEP 4:

For each particle i (i=1,2,…,*Ns*) calculate its new position as in the following:

$$x_{i,k} = x_{i,k-1} + v_{i,k}, \qquad \text{for } k>0 \tag{8}$$

STEP 5:

Repeat steps (1)-(4) until k exceeds a maximum (user-defined) number of iterations.

4. The proposed PSO algorithm for the CDD SMETSP

4.1 Solution space and encoding mechanisms

The core idea behind the developed PSO algorithm is to search for solutions that are V-shaped. This is accomplished by the following encoding scheme that designates each job either being early or tardy: assuming an *n*-job SMETSP a candidate solution $x_{i,k}$ (i=1,2,…,*Ns*) denoting the position of the i-th particle in

the k-th iteration, is a vector containing n floating-point numbers taken from values within the range [0,1]. Each such floating-point number is associated to a specific job 1,2,…,n with that order. A value less than or equal to 0.5 in the vector indicates that the corresponding job is early otherwise the job is tardy. So, the jobs are distinguished into two sets namely, S_E and S_T containing the early and the tardy jobs respectively. Following the V-shaped property, the jobs in S_E are moved in the start of the schedule and sequenced in '\-shaped' format. Late jobs are moved in the end of the schedule and sequenced in '/-shaped' format. Let *sump* the total processing time of the early jobs in S_E, then an optimal solution to SMETSP can fall in one of the following three cases[18]:

(A) The first job in S_E starts at time zero and the last job in S_E finished exactly on due date d.
(B) The first job in S_E starts at time zero and the last job in S_E is completed prior to d. Further, a straddling job exists, i.e., a job starting executed before d and ending after d.
(C) The first job in S_E does not start at time zero (i.e., it is delayed) and the last job in S_E is finished exactly on the due date d.

Case-(A) occurs when $sump=d$; case-(B) occurs when $sump>d$, while case-(C) occurs when $sump<d$. Therefore, according to the proposed scheme, for every candidate vector of the entire population, the sets S_E and S_T are firstly created. Second, the processing time of the jobs in S_E are summed up into *sump* until the value of this variable surpassed d or no other jobs are contained in S_E. Third, the starting time of the first job in S_E is defined. That is, when $sump \geq d$ (cases (A) and (B)), the first job starts at time zero, otherwise (case (C)), the first job is delayed starting at time $d-sump$. Fourth, jobs in S_E are ordered based on '\-shaped' property, while jobs in S_T are ordered based on '/-shaped' property. This encoding mechanism is given below in algorithmic form. $\Psi=(\psi_{1,\ldots,}\psi_n)$ is an individual vector solution, while the notation $\{i\}$ denotes the i-th job (i=1,…,n).

Procedure Encoding_mechanism (Ψ)
begin
Step1: Build sets S_E and S_T
$S_E = S_T = \varnothing$; *sump* = 0 ; caseB = **false** ;
for i = **1 to** n **do**
if ($\psi_i \leq 0.5$) **then**
if ($sump + p_{\{i\}} \leq d$) **then**
$sump = sump + p_{\{i\}}$;
$S_E = S_E \cup \{i\}$; // insert $\{i\}$ into S_E //
else if not caseB **then**
caseB = **true** ;
$k = \{i\}$; // k is the straddling job //
else $S_T = S_T \cup \{i\}$; // insert $\{i\}$ into S_T //
endif
else $S_T = S_T \cup \{i\}$;
endif
endfor
Step2: Build the final schedule
if caseB **then** // case-(B) //
Sort the jobs in S_E according to '\-shaped' format. Then, put at the tail of S_E the straddling job k. Sort the jobs in S_T according to '/-shaped' format. First job in S_E starts at time *Tstart* = 0.
else // case-(A), or -(C) //
Sort the jobs in S_E *and* S_T according to '\/-shaped' property. First job in S_E starts at time
Tstart = d - *sump*
endif
$g = S_E + S_T$ // final schedule //
Return (g, *Tstart*)
end

Let us discuss how this mechanism works through a simple example for the 8-job SMETSP given in Table 1. The demand is to finish the jobs on a common due date d=55. The summation of the tasks' processing times is equal to $\sum p_i$=91 (i=1,…,8). Suppose the following floating-point vector is given, generated at some point in time by PSO algorithm

Ψ = (0.33, 0.76, 0.10, 0.40, 0.05, 0.20, 0.11, 0.86)

Table 1: Jobs' characteristics for an 8-jobs SMETSP.

	j1	j2	j3	j4	j5	j6	j7	j8
p_i	20	6	13	13	12	12	12	3
α_i	4	1	5	2	7	9	5	6
β_i	5	15	13	13	6	6	15	1
p_i/α_i	5	6	2.6	6.5	1.7	1.3	2.4	0.5
p_i/β_i	4	0.4	1	1	2	2	0.8	3

Since the 2nd and 8th components of Ψ have values greater than 0.5, then jobs 2 and 8 are put in the tardy

Co-published by Atlantis Press and Taylor & Francis

set S_T. All the remaining jobs are candidate members of the early set S_E. However, a job $\{i\}$ is placed in S_E only when the following three conditions are satisfied: (1) $\psi_i \leq 0.5$, (2) $sump + p_1 \leq d$, and (3) $\{i\}$ is not a straddling job. Otherwise, $\{i\}$ is placed in S_T.

Hence, after applying step 1 of the above encoding mechanism, the two sets are formed as S_E={1, 3, 4}, and S_T={2, 6, 7, 8}; while job {5} becomes a straddling job. Therefore, the solution represented by Ψ falls into case-(B), meaning that the jobs in S_E and S_T must be sequenced according to '\-shaped' and '/-shaped' order, respectively. While the straddling job must be put between the tail of S_E and the head of S_T. Using the values given in Table 1, the final V-shaped solution corresponding to Ψ is displayed in Figure 2.

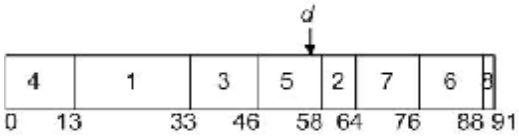


Fig. 2: The schedule for the 8-job problem of Table 1 corresponding to the particle

4.2 PSO implementation for the SMETSP

Except from the above encoding mechanism which is necessary for mapping a particle's position to an actual schedule solution, in order to apply PSO to SMETSP a way must be found for using Eqs (6) and (8). To that purpose, a technique similar to that previously used by Allahverdi and Al-Anzi[43] for the solution of a two-stage assembly-scheduling problem was adopted. Let us see how this technique works through a simple example:

Suppose that, for a 5-job SMETSP at some point in time during the *k*-th iteration, a specific particle say $x_{j,k}$ ($\varphi \in [1, Ns]$) in the swarm has the following characteristics: *Present* = (3, 4, 1, 2, 5), *Pbest* = (1, 4, 5, 2, 3), and *Gbest* = (2, 4, 5, 1, 3). *Present* denotes the SMETSP solution represented by $x_{j,k}$. *Pbest* denotes the schedule solution corresponding to $x_{f,k}^{pbest}$ and *Gbest* the schedule solution corresponding to x_k^{gbest}.

Let, D_1 be the fraction of jobs that are different between *Present* and *Pbest*; and D_2 the fraction of jobs that are different between *Present* and *Gbest*. In other words, D_1 and D_2 represent the differences between the jobs' sequences *Present* and *Pbest*, and *Present* and *Gbest*, respectively. Therefore, for the above example, since *Present* differs from *Pbest* in three locations (actually in the 1st, 3rd, and 5th locations) then D_1=3/5= 0.6. Similarly, *Present* differs from *Gbest* in four (out the five in total) locations meaning that D_2=4/5 = 0.8. Eqs. (6) and (8) can now be written as in the following[43]:

Advance *Present* towards *Pbest* with a velocity,

$$V_1 = c_1 \cdot r_1 \cdot D_1 \tag{9}$$

Advance *Present* towards *Gbest* with a velocity,

$$V_2 = c_2 \cdot r_2 \cdot D_2 \tag{10}$$

r_1, r_2 are uniform random numbers drawn in (0,1). Advancing *Present* towards *Pbest* at a speed equal to V_1 is implemented using the procedure:

Procedure Move_towards_*Pbest* ($x_{j,k}$, *Present*, *Pbest*, $x_{f,k}^{pbest}$, V_1)
begin
  for j =1 **to** *Ns* **do**
    if $Present_j \neq Pbest_j$ **then**
      if $R^j \leq V_1$ **then**
        $x_{f,k}^{j} = x_{f,k}^{pbest,j}$ // update particle's position //
      endif
    endif
  endfor
  Return ($x_{j,k}$)
end

Where, $Present_j$ denotes the j-th job in *Present* sequence, and $Pbest_j$ the j-th job in *Pbest* sequence. $R^j \in (0,1)$, is drawn randomly for each j. Similarly, advancing *Present* towards *Gbest* at a speed equal to V_2 is implemented using the procedure:

Procedure Move_towards_*Gbest* ($x_{j,k}$, *Present*, *Gbest*, x_k^{gbest} , V_2)
begin
  for j =1 **to** *Ns* **do**
    if $Present_j \neq Gbest_j$ **then**
      if $R^j \leq V_2$ **then**
        $x_{f,k}^{j} = x_k^{gbest,j}$ // update particle's position //

endif
endif
endfor
Return ($x_{j,k}$)
end

Therefore, if the job lying at position j (j=1,…,n) of *Present* is different from the associated job in *Pbest* (*Gbest*), then make the content of the j-th component of $x_{j,k}$ same to that of the j-th component of x_k^{pbest} (x_k^{gbest}) with probability V_1 (V_2). Note that, V_1 and V_2 are limited to take values in the range (0,1). Possible negative values are set to 0.01, and values greater than 1 are set to 0.95. The complete version of the proposed PSO algorithm is given below:

Algorithm PSO for the SMETSP
Input: The number of the jobs to be scheduled (n). The common due date (d). Three quantities (p_j , a_j , b_j) ∀ job j (j=1,…,n) corresponding to the job's processing time, and job's earliness and tardiness penalties, respectively.
Output: The 'best' V-shaped Schedule (VS*).
Begin
Step 1: Initialize control parameters
1.1) Set the size *Ns* of the swarm;
1.2) Set values for the weight factors c_1 and c_2;
1.3) Initialize iteration counter k = 0, and set the maximum number of iterations k_{MAX};
Step 2: Swarm Initialization
2.1) Set $x_{\min}$=0 , $x_{\max}$ =1, $v_{\min}$ =0, $v_{\max}$ =1;
2.2) Build initial swarm S using Eq. (3).
2.3) For each particle i (i=1,2,…,*Ns*) in S, create its initial velocity $v_{i,k}$ using Eq. (4).
Step 3: Swarm Evaluation
for i = 1 to *Ns* **do**
3.1) Build schedule $vs^{x_{i,k}}$ corresponding to $x_{i,k}$
3.2) **Set** $vs^{x_{i,k}}$ = *Encoding_mechanism*($x_{i,k}$);
3.3) Compute the cost of the generated schedule, *COST*($vs^{x_{i,k}}$) using Eq. (1);
3.4) Determine the personal best position and its associated V-shaped schedule as in the following:
if (k=0) **then**
$x_{i,k}^{pbest} = x_{i,k}$; $vs_{i,k}^{pbest} = vs^{x_{i,k}}$;
elseif *COST*($vs^{x_{i,k}}$) < COST($vs_{i,k}^{pbest}$) **then**
$x_{i,k}^{pbest} = x_{i,k}$; $vs_{i,k}^{pbest} = vs^{x_{i,k}}$;
endif
endfor
Step 4: Population Statistics:
4.1) Determine the global best V-shaped schedule of the swarm (vs_k^{gbest}); save the associated global best position into x_k^{gbest} ;
4.2) **Set** *Gbest* = vs_k^{gbest} ;
4.3) Keep track for the *best-so-far* V-shaped solution vs^* :
if k=0 **then** $vs^* = vs_k^{gbest}$
else if COST(vs_k^{gbest}) < COST(vs^*) **then**
$vs^* = vs_k^{gbest}$
endif
Step 5: Update velocity and position
for each particle i (i=1,2,…,*Ns*) in S **do**
5.1) **Set** *Present* = $vs^{x_{i,k}}$, *Pbest* = $vs_{i,k}^{pbest}$;
5.2) Determine D_1 and D_2 using the method described in sub-section 4.2;
5.3) Compute V_1 using Eq.(9), repair V_1 if needed;
5.4) Compute V_2 using Eq.(10), repair V_2 if needed;
5.5) Advance *Present* towards *Pbest* at a speed V_1 using: Move_towards_*Pbest*($x_{i,k}$,*Present, Pbest,* $x_{i,k}^{pbest}$, V_1);
5.6) Advance *Present* towards *Gbest* at a speed V_2 using:Move_towards_*Gbest* ($x_{i,k}$, *Present, Gbest,* x_k^{gbest} , V_2) ;
endfor
Step 6: Stopping criterion
6.1) Advance iteration counter: $k = k + 1$;
6.2) if $k \le k_{MAX}$ then go to Step 3.
6.3) Return (vs^*);
End;

5. Experimental analysis and discussion

The proposed PSO algorithm was implemented in Pascal and run on a Pentium 4 (1.2 GHz) PC. The algorithm was tested over a set of public benchmarks problems, recently proposed by Biskup and Feldmann[25]. These benchmarks include test instances ranging from small size with 10 jobs to large size instances with 1000 jobs. Specifically, there are seven categories of problems with 10, 20, 50, 100, 200, 500, and 1000 number of jobs with each category containing ten instances to be tested. The value of a restrictive

Co-published by Atlantis Press and Taylor & Francis

factor h=0.2, 0.4, 0.6, 0.8 classifies the problems as less or more restricted against a common due date d using the relation:

$$d = \left\lfloor h \sum_{j=1}^{n} p_j \right\rfloor \quad (11)$$

With $\lfloor y \rfloor$, denoting the biggest integer smaller than, or equal to y. That is, for each problem, the common due date d is estimated by multiplying the summation of the processing times of all the n jobs with the restrictive factor h. The lower the value of h the more restrictive is d (the higher the expected percentage of the late jobs). Note that h must be within (0, 1) recognizing that, for h=1 the problem becomes unrestrictive, while for h=0 the problem becomes trivial. It is worth pointing out that optimal solutions for the examined benchmarks only exist for the 10 jobs test instances and have been achieved using an integer programming formulation with LINDO software[25].

As described in section 3 the performance of the proposed PSO algorithm is depended on the values of three parameters: the swarm size (Ns), the cognitive parameter (c_1), and the social parameter (c_2). In order to determine the correct settings for these parameters, the following control schemes were studied:

(a) Different swarm sizes were examined in the range, $Ns \in \{max(10, 5+\lfloor\sqrt{n}\rfloor), n, 2n, 3n\}$; with n *being the* number of the jobs to be scheduled. The function max(x,y) returns the maximum between x and y. Note that, formula $max(10, 5+\lfloor\sqrt{n}\rfloor$ guarantees that Ns will be never less than 10 jobs (cases of test problems with n=10 and n=20 jobs). Using swarms with less than 10 particles was found in preliminary experiments to reduce drastically the performance of the developed algorithm resulting to solutions with poor quality.

(b) To be consistent with the literature[46,47] the two learning factors c_1 and c_2 were both set to a fixed and equal value within the range {0.5, 1.0, 2.0}. Furthermore, since some recent works (e.g., see Refs. 47 and 48) report that it might be even better to choose a larger cognitive parameter, c_1, than a social parameter, c_2, but with $c_1+c_2 \leq 4$, experiments have been also conducted using the combination c_1=2 and c_2=1.5.

In all the simulations, the PSO algorithm was left running for a maximum number of 500n evaluations (where an evaluation corresponds to the estimation of the objective (cost) function of an individual solution) and the best results obtained after this time duration were reported. The particular maximum running duration was found in preliminary experiments to be a quite good choice for terminating the running of the developed algorithm. It was the main result of an empirical investigation performed with the following 5 termination conditions alternatives: stop after running for a maximum of, (a) 500n evaluations, (b) 1000n evaluations, (c) 1000 iterations; and (d) stop the algorithm if after a number of 50 iterations no improvement to the best so far solution is encountered.

Moreover, in order to be fair with the stochastic behavior of the algorithm, multiple runs over each test instance were performed and the average results were reported. More specifically, to get the average performance of the PSO algorithm, 15 runs (starting each time from a different random number seed) on each problem instance were performed and the solution quality was averaged. Two performance measures were used to quantify the performance of the algorithm:

(i). The average percentage deviation (**dev%**) from the existing optimum solution defined by the equation:

$$dev\% = 100 \times \left(Cost_{PSO} - UB \right) / UB \quad (12)$$

Where, $Cost_{PSO}$ is the cost (Eq.(1)) of the best schedule achieved by the PSO algorithm for a specific benchmark instance; and UB (=upper bound) the corresponding cost of the existing best known solution for this benchmark instance as reported in Ref. 25.

(ii). The average CPU time (measured in seconds) consumed over each test instance until the convergence of the algorithm.

Table 2 displays the influence of the various combinations of settings of the control parameters (Ns, c_1, c_2) on the performance of the PSO algorithm in regard to *dev%*. The results displayed concern the application of the PSO algorithm over the 10 instances included in the 50-job SMETSP (h=0.2) benchmarks. It is underlined that a similar influence of these

parameters on the performance of the algorithm was also investigated over the other benchmarks categories. As one can see from Table 2, best results were encountered using the combination, $Ns=max(10, 5+\lfloor\sqrt{n}\rfloor)$, c_1=2.0, and c_2=1.5. The minus sign in the numerical results of this table denotes that PSO attained to generate schedules of lower costs than the costs of the existing best schedules for the specific benchmarks. Particularly, PSO improved the existing best objective function values for these benchmarks by approx. 5.5%.

Table 2: Choosing the correct control settings for the PSO algorithm. Average dev% of the best solutions obtained after 15 runs over the instances of the 50-jobs (h=0.2) benchmarks.

Ns	$c_1=c_2$ = 0.5	$c_1=c_2$ = 1	$c_1=c_2$ = 2	c_1=2.0 c_2=1.5
$max(10, 5+\lfloor\sqrt{n}\rfloor)$	-5.21	-5.42	-5.48	**-5.55**
n	-5.29	-5.26	-5.26	-5.34
$2n$	-4.98	-5.18	-5.18	-5.21
$3n$	-5.02	-5.14	-5.20	-5.20

In regard to the results of Table 2, one more comment about the selection of the appropriate population (swarm) size must be given for the interested reader. The empirical knowledge generally dictates that a larger population will work more slowly but will eventually achieve better solutions than a smaller population. However, this is not always true and the correct population size depends on the problem being solved, the run duration of the algorithm, the representation used and the operators manipulating this representation[49]. The experimental results presented throughout this paper are performed with the PSO algorithm running for a maximum number of 500n evaluations. Where, an evaluation corresponds to the estimation of the objective function of an individual solution. That is, given that the PSO algorithm is going to perform 500n=500×50=25,000 evaluations for the 50-job SMETSP, 50,000 evaluations for the 100-job SMETSP, and so on, the above preliminary tests indicated that a population of $max(10, 5+\lfloor\sqrt{n}\rfloor)$ individuals is an effective choice to deal with the specific implementation of the PSO algorithm.

The following discussion concerns the application of the PSO algorithm using the above 'best' combination of settings. The experiments have been carried out over the most difficult test instances of the Biskup and Feldmann (2001)'s benchmarks[25]; i.e., those instances that are more restricted against common due dates (i.e., with h = 0.2 and 0.4). The full computational results obtained by the proposed PSO algorithm are summarized in Tables 3 and 4. In particular, Table 3 contains the results concerning the small size benchmark problems with up to 50 jobs; while Table 4 contains the results concerning the large size benchmarks problems with jobs ranging from 100 to 1000. For each test instance the two tables include the existing upper bound (*UB*) generated by Biskup and Feldmann[25], the near-optimum solution achieved by the PSO algorithm ($Cost_{PSO}$), and the percentage offset (*dev%*) of the generated solution from the existing upper bound. It is worth pointing out that optimal solutions for these benchmarks only exist for the 10 jobs test instances and have been achieved using an integer programming formulation with LINDO software[25].

Table 3: Results over the small size benchmarks.

		***h*=0.2**			***h*=0.4**		
n		UB/ OPTIMUM	$Cost_{pso}$	dev%	UB/ OPTIMUM	$Cost_{pso}$	dev%
10	1	1936	1936	0.00	1025	1025	0.00
	2	1042	1042	0.00	615	615	0.00
	3	1586	1602	1.01	917	931	1.53
	4	2139	2169	1.40	1230	1230	0.00
	5	1187	1187	0.00	630	630	0.00
	6	1521	1521	0.00	908	908	0.00
	7	2170	2170	0.00	1374	1374	0.00
	8	1720	1720	0.00	1020	1020	0.00
	9	1574	1574	0.00	876	876	0.00
	10	1869	1869	0.00	1136	1140	0.35
20	1	4431	4394	-0.84	3066	3066	0.00
	2	8567	8460	-1.25	4897	4897	0.00
	3	6331	6221	-1.74	3883	3845	-0.98
	4	9478	9192	-3.02	5122	5122	0.00
	5	4340	4215	-2.88	2571	2495	-2.96

Co-published by Atlantis Press and Taylor & Francis

	6	6766	6552	-3.16	3601	3584	-0.47
	7	11101	10459	-5.78	6357	6250	-1.68
	8	4203	3994	-4.97	2151	2201	2.32
	9	3530	3465	-1.84	2097	2096	-0.05
	10	5545	4987	-10.06	3192	2925	-8.36
50	1	42363	40734	-3.85	24868	23792	-4.33
	2	33637	30739	-8.62	19279	18042	-6.42
	3	37641	34505	-8.33	21353	20700	-3.06
	4	30166	27803	-7.83	17495	16693	-4.58
	5	32604	32332	-0.83	18441	18167	-1.49
	6	36920	35102	-4.92	21497	20402	-5.09
	7	44277	43229	-2.37	23883	23228	-2.74
	8	46065	43969	-4.55	25402	24947	-1.79
	9	36397	34326	-5.69	21929	20008	-8.76
	10	35797	32999	-7.82	20048	19238	-4.04

Table 4: Results over the large size benchmarks.

		***h*=0.2**			***h*=0.4**		
n		UB/ OPTIMUM	$Cost_{pso}$	*dev%*	UB/ OPTIMUM	$Cost_{pso}$	*dev%*
100	1	156103	146132	-6.39	89588	86280	-3.69
	2	132605	125331	-5.49	74854	73459	-1.86
	3	137463	130414	-5.13	85363	80207	-6.04
	4	137265	131132	-4.47	87730	79947	-8.87
	5	136761	124882	-8.69	76424	71609	-6.30
	6	151938	139961	-7.88	86724	77987	-10.07
	7	141613	137407	-2.97	79854	78410	-1.81
	8	168086	161424	-3.96	95361	95612	0.26
	9	125153	118859	-5.03	73605	69812	-5.15
	10	124446	119795	-3.74	72399	72389	-0.01
200	1	526666	502920	-4.51	301449	298080	-1.12
	2	566643	556476	-1.79	335714	322705	-3.88
	3	529919	497396	-6.14	308278	296023	-3.98
	4	603709	599074	-0.77	360852	356197	-1.29
	5	547953	524601	-4.26	322268	306819	-4.79
	6	502276	490165	-2.41	292453	287009	-1.86
	7	479651	470485	-1.91	279576	275805	-1.35
	8	530896	505432	-4.80	288746	280383	-2.90
	9	575353	545248	-5.23	331107	315441	-4.73
	10	572866	544871	-4.89	332808	327299	-1.66
500	1	3113088	2995837	-3.77	1839902	1809150	-1.67
	2	3569058	3396297	-4.84	2064998	2019880	-2.18
	3	3300744	3114316	-5.65	1909304	1877994	-1.64
	4	3408867	3243457	-4.85	1930829	1923249	-0.39
	5	3377547	3127020	-7.42	1881221	1824148	-3.03
	6	3024082	2807257	-7.17	1658411	1637076	-1.29
	7	3381166	3193413	-5.55	1971176	1931460	-2.01
	8	3376678	3127525	-7.38	1924191	1829059	-4.94
	9	3617807	3376250	-6.68	2065647	1995866	-3.38
	10	3315019	3137808	-5.35	1928579	1867701	-3.16
1000	1	15190371	14496459	-4.57	8570154	8165112	-4.73
	2	13356727	12637411	-5.39	7592040	7324309	-3.53
	3	12919259	12424534	-3.83	7313736	7057780	-3.50
	4	12705290	12332609	-2.93	7300217	7063524	-3.24
	5	13276868	13013517	-1.98	7738367	7401864	-4.35
	6	12236080	11683666	-4.51	7144491	7012026	-1.85
	7	14160773	13299602	-6.08	8426024	7898469	-6.26
	8	13314723	12321242	-7.46	7508507	7264332	-3.25
	9	12433821	11789909	-5.18	7299271	7117968	-2.48
	10	13395234	12471333	-6.90	7617658	7330962	-3.76

As one can see from Tables 3 and 4 the performance of the PSO algorithm is of high quality. The algorithm reached the exact optimum solution in the majority of the test instances with 10-jobs (see Table 3); particularly, in the eight out the ten in total instances. Furthermore, (except from three instances of the h=0.4, 20-job benchmarks –see Table 3-) higher quality solutions than the existing best known solutions have been generated by PSO to all the test instances ranging from 20 to 1000 jobs. This fact is indicated by a negative percentage deviation (dev%) in the corresponding columns of the two tables. More specifically, for the most restricted class of benchmarks (h=0.2) the mean dev% for the solutions obtained are -3.55% for 20 jobs problems, -5.48% for 50-jobs problems (Table 3), -5.38% for 100-jobs, -3.67% for 200-jobs (Table 4), and so on. PSO introduced new upper bounds in all the h=0.2 instances with n≥20, i.e. to approx. 86% of the instances of h=0.2 benchmarks. While, in regard to the h=0.4 benchmarks, PSO attained

to improve nearly the 75% of the available test instances. Figure 3 illustrates the average percentile improvement of the benchmarks solutions attained by the PSO algorithm.

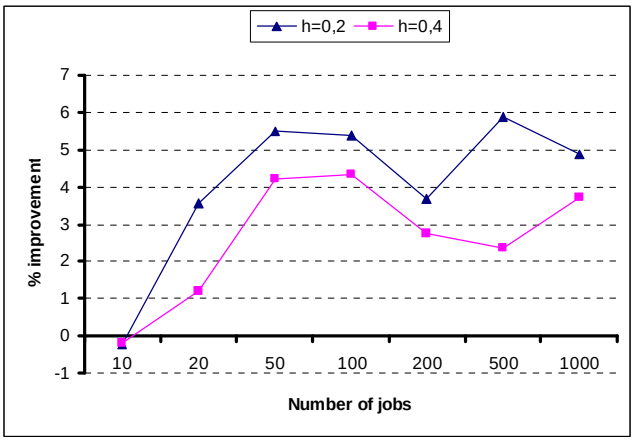
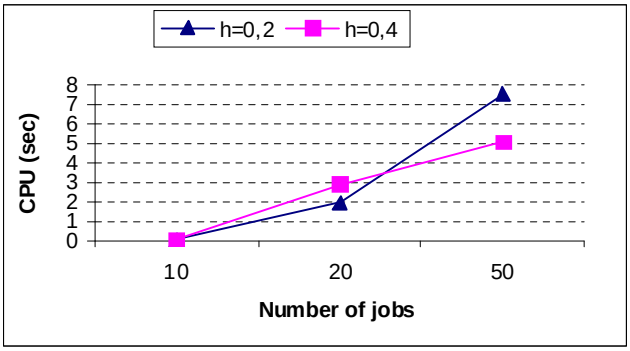
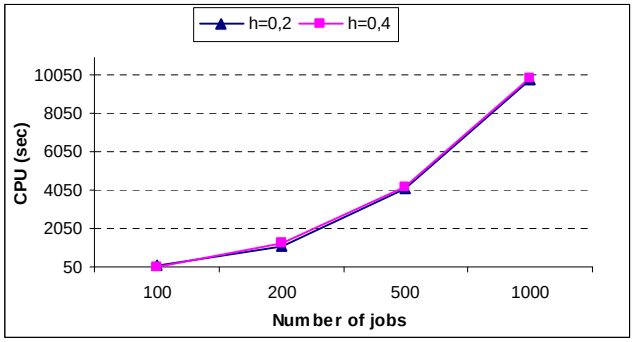


Fig. 3: Average percentile improvement of the benchmarks solutions using the PSO algorithm.

Figure 4 displays the mean actual running times for the PSO algorithm until the convergence to the near-optimum solution. The running times (in seconds) are given in regard to the problems' degree of restriction (value of parameter *h*) and problems' sizes.



(a)



(b)

Fig. 4: Average CPU times (in seconds) until the convergence. Results on a Pentium 4 (1.2 GHz) PC. (a) case of small size problems, (b) case of large size problems.

Figure 5 displays the optimal solutions (in Gantt charts) obtained by PSO for the first two instances included in the *h*=0.2 10-jobs benchmarks problems. f* denotes the cost (given by Eq. (1)) of the optimal solution. From Table 3 one can observe that PSO did not find the exact optimum solution in 4 (out the 20 in total) instances of the 10-jobs test beds. However, it is worth pointing out that, with a slight modifications of the swarm's size (particularly, with a swarm's size equal to 4*n* which in the case of the 10-jobs problems means 40 particles in the swarm), PSO attained to determine the exact optimal solution for these 4 instances too.

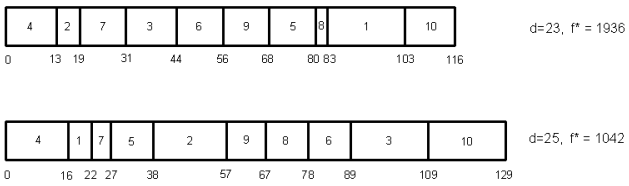
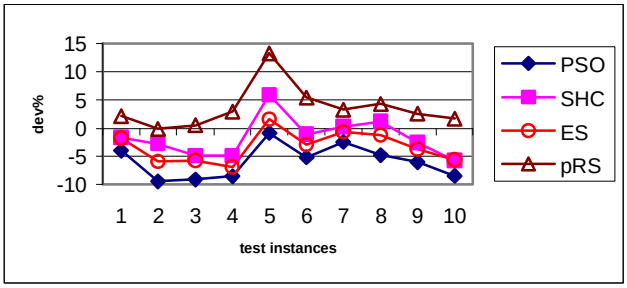


Fig. 5: The Gantt charts for the optimal solutions obtained by PSO for the first two instances included in 10-jobs (h=0.2) benchmarks problems. d and f* denotes the common due-date and the cost of the optimal solution, respectively.

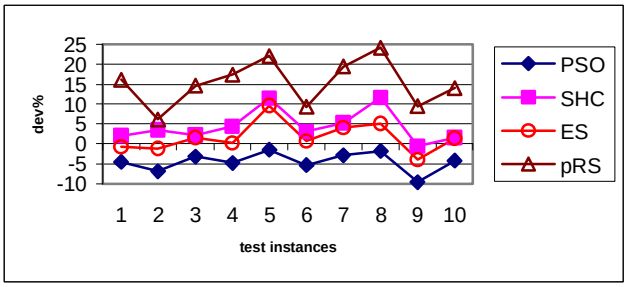
Some interesting comparative results between PSO and three known heuristics namely, evolution strategy (ES), stochastic hill-climber (SHC), and pure random search (pRS) technique are depicted in Fig. 6. These

Co-published by Atlantis Press and Taylor & Francis

results concern the dev% of the solutions obtained by the four methods (averaged after 15 runs) on the test instances included in the 50-jobs data sets. As one can observe from this figure, PSO (lower curve) outperforms all the other heuristics generating solutions of much higher quality (schedules of lower costs). As it was expected, the next best performance was due to ES, while the worst performance was due to pRS technique. The basic characteristics of the three implemented heuristics are briefly described below.



(a)



(b)

Fig. 6: Comparative results between PSO, ES, SHC and pRS over the 10 test instances included in the 50-jobs benchmarks data set. The % deviation of the solutions obtained by the four heuristics from the existing near-optimal benchmark solutions. Case of, (a) h=0.2 and (b) h=0.4 tests instances.

To make the three heuristics comparable to PSO we used the same population size for all, and limit the running process for the same maximum number of iterations. Moreover, all the heuristics were implemented using the (same with PSO) encoding, and evaluation mechanisms described in sub-section 4.1. Particularly, each individual solution in ES, SHC and pRS is a vector containing n floating-point numbers within the range [0,1]. Each such floating-point vector is mapped to an actual SMETSP solution using procedure *Encoding_mechanism* also described in sub-section 4.1. ES was implemented by following the recommendations of the literature[50]. After experimentation for choosing the correct survivor selection scheme, we found that using (μ,λ)-scheme within ES results in a superior optimizer to that of using $(\mu+\lambda)$-scheme. For this reason (μ,λ)-scheme was adopted. μ was estimated using the heuristic rule $\lambda/\mu=7$. Mutation was performed through Gaussian perturbation. Offspring were generated using discrete recombination. SHC was implemented as a modified version of the metropolis algorithm[51] for a parallel searching of the solution space. That is, for each solution $x_{i,k} \in S$ in iteration k, search its neighbourhood $N(x_{i,k})$ and choose the best solution $x'_{i,k}$. Accept this solution (i.e., $x_{i,k}=x'_{i,k}$) if COST($x'_{i,k}$)<COST($x_{i,k}$) or with a probability $exp((\text{COST}(x_{i,k})-\text{COST}(x'_{i,k}))/T)$. $T>0$ (called temperature) determines the probability of accepting worse solutions. A value of $T=10$ was experimentally found to be a good choice for our problem. A new solution in $N(x_{i,k})$ was generated by flipping randomly each time the j^{th} ($j=1,\ldots,n$) component of $x_{i,k}$. Thus, for each solution we generate n different neighbours. pRS is identical to SHC except from the point that, a new solution $x'_{i,k}$ is accepted and replaced its ancestor $x_{i,k}$ only if it has a lower cost.

We conclude by giving a short guide to those of the researchers who are willing to apply PSO on their discrete COPs: (a) Select an appropriate representation. This representation is directly depended on the structure of the problem being solved. (b) Design a suitable encoding mechanism that maps each vector solution to an actual COP's solution. This mechanism must take into account the properties of the problem being solved. (c) If necessary, implement appropriate repairing mechanisms to guarantee the creation of legal solutions. (d) Follow the recommendation of the literature to estimate the proper settings of the control parameters (i.e., swarm size, inertia weight, learning factors). (e) Study the behavior of your PSO at different iterations so that to realize whether it converges too early. There are some important tricks in the literature to improve the rate of convergence of PSO, to control the diversity of the swarm so that to avoid local optima, etc.

6. Conclusion

In today production management, scheduling against common due dates with respect to earliness and tardiness penalties is of great importance mainly due to its compliance with the principles of just-in-time 'philosophy'. This is a typical discrete combinatorial optimization problem (COP) known to be intractable, meaning that, the search for optimal schedules using traditional mathematical programming techniques is only possible for very small size instances of the problem.

This paper presented a new particle swarm optimization (PSO) heuristic to address the problem of scheduling a number of jobs on a single machine against a restricted common due date. It is worth noting that PSO-based heuristics have rarely been applied to discrete COPs. The developed PSO differs from the existing PSO approaches in two main points: first in the way the 'particles' are represented and mapped into actual scheduling solutions; and second, in the way of controlling the 'particles' velocities of the entire swarm. Extensive experiments were performed over 140 highly restricted against a common due date benchmarks problems for which the upper bounds were known. The results obtained showed a high quality performance for the proposed PSO heuristic introducing to the majority of the benchmark instances new upper bounds. In particular, PSO reached or surpassed the 95% of the existing optimal benchmark solutions introducing new upper bounds in the 82% of the benchmarks instances. Furthermore, comparative experiments with existing heuristics (including evolution strategy, stochastic hill-climber, and random search) showed a substantially superior performance for the developed approach in terms of solution quality. The method is quite general and can be rather easily modified and applied to any other COP. The only modifications required to be done is, firstly in the encoding mechanism (that is mapping the particles to actual solutions to the considered problem), and secondly in the evaluation mechanism (i.e. the objective function which is problem depended). Moreover, due to the small number of control parameters used within the algorithm, parameters' tuning is performed easily. A limitation however of the proposed method is its convergence speed. In its current form the algorithm is rather slow when the number of the jobs to be scheduled exceeds 200.

Future work will be focused on the development of a more robust and faster version of the PSO algorithm to address other more difficult scheduling problems that are known to be intractable, such as the job-shop scheduling problems. Hybridization of the PSO algorithm with local search techniques such as tabu-search or/and simulated annealing is a very promising area of research and may result in more powerful heuristic. On going research is focused to the design of a multi-objective PSO algorithm to address the multi-criteria flow-shop scheduling problem. Three criteria are currently under investigations with the objective to be simultaneously minimized, namely, the makespan, the maximum tardiness and the total flow time of the jobs.

References

1. K. Baker and G. Scudder, Sequencing with earliness and tardiness penalties: A review, *Operations Research* 38 (1990) 22-36.
2. S. French, *Sequencing and scheduling, an introduction to the mathematics of the job-shop* (Ellis Horwood publication, 1990).
3. J.J. Kanet, Minimizing the average deviation of job completion times about a common due date, *Naval Research Logistics* 28 (1981) 643-651.
4. N. Hall, Single and multiple processor models for minimizing completion time variance, *Naval Research Logistics Quarterly* 33 (1986) 49-54.
5. U. Bagchi, R.S. Sullivan, and Y.L Chang. Minimizing mean absolute deviation of completion times about a common due-date, *Naval Research Logistics Quarterly* 33 (1986) 227-240.
6. U. Bagchi, R.S. Sullivan, and Y.L. Chang, Minimizing absolute and squared deviations of completion times with different earliness and tardiness penalties about a common due-date, *Naval Research Logistics Quarterly* 34 (1987) 739-751.
7. N. Hall and M. Posner, Earliness-tardiness scheduling problems I: weighted deviation of completion times about a common date, *Operations Research* 39 (1991) 836-846.
8. P. De, J.B Ghosh, C.E. Wells, Solving a generalized model for CON due date assignment and sequencing, *Int. Journal of Production Economics* 34 (1994) 179-185.
9. J.A. Hoogeveen, and S.L. van de Velde, Earliness-tardiness scheduling around almost equal due date, *INFORMS Journal on Computing* 9 (1997) 92-99.

Co-published by Atlantis Press and Taylor & Francis

10. T. Cheng and M. Gupta, Survey of scheduling research involving due date determination decision, *European Journal of Operational Research* 38 (1989) 156-166.
11. V. Gordon, J.-M.Proth, and C. Chu, A survey of the state-of-the-art of common due date assignment and scheduling research, *European Journal of Operational Research* 139 (2002) 1-25.
12. T. Abdul-Razaq and C.N. Potts, Dynamic programming state-space relaxation for single-machine scheduling, *Journal of the Operational Research Society* 39 (1988) 141-152.
13. J.P. Souza and L.A. Wolsey, A time indexed formulation of non-preemptive single machine scheduling problems, *Mathematical Programming* 54 (1992) 353-367.
14. M.-T. Almeida and M. Centeno, A composite heuristic for the single machine early/tardy job scheduling problem, *Computers and Operations Research* 25 (1998) 625-635.
15. C.-Y. Lee and S.J. Kim, Parallel genetic algorithms for the earliness-tardiness job scheduling problem with general penalty weights, *Computers and Industrial Engineering* 28 (1995) 231-248.
16. Q. Hao, Z. Yang, D. Wang, Z. Li, Common due date determination and sequencing using tabu search, *Computers and Operations Research* 23 (1996) 409-417.
17. R.J.W. James, Using tabu search to solve the common due date early/tardy machine scheduling problem, *Computers and Operations Research* 24 (1997) 199-208.
18. M. Feldmann and D. Biskup, Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches, *Computers and Industrial Engineering* 44 (2003) 307-323.
19. A.C. Nearchou and S.L. Omirou, Differential evolution for sequencing and scheduling optimization, *Journal of Heuristics* 12 (2006) 395–411.
20. A.C. Nearchou, A differential evolution approach for the common due date early/tardy job scheduling problem, *Computers & Operations Research* 35 (2008) 1329-1343.
21. Z.-J. Lee, C.-C. Chuang and K.-C. Ying, An intelligent algorithm for scheduling jobs on a single machine with a common due date, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4693 LNAI (PART 2) (2007) 689-695.
22. Z.-J. Lee, S.-W. Lin, K.-C. Ying, A dynamical ant colony optimization with heuristics for scheduling jobs on a single machine with a common due date, *Studies in Computational Intelligence* 128 (2008) 91-103.
23. R. M'Hallah and A. Alhajraf, Ant Colony Optimization for the Single Machine Total Earliness Tardiness Scheduling Problem, in *New Frontiers in Applied Artificial Intelligence: Lecture Notes in Computer Science*, N.T. Nguyen *et al.* (Eds.) 5027 (2008) 397-407.
24. M. Reisi and G. Moslehi, Minimizing the number of tardy jobs and maximum earliness in the single machine scheduling using an artificial immune system, *Int Journal of Advanced Manufacturing Technology* 54 (2011) 749–756
25. D. Biskup and M. Feldmann, Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates, *Computers and Operations Research* 28 (2001) 787-801.
26. R.C. Eberhart and J. Kennedy, A new optimizer using particle swarm theory, *in Proc. of* the 6th *Int. Symposium on Micro Machine and Human Science*, (1995) 39-43.
27. Anghinolfi and M. Paolucci, A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times, *European Journal of Operational Research* 193 (2009) 73-85.
28. C. Low, C.-J. Hsu, C.-T. Su, A modified particle swarm optimization algorithm for a single-machine scheduling problem with periodic maintenance, *Expert Systems with Applications* 37 (2010) 6429-6434.
29. Q.-K. Pan, M.F. Tasgetiren and Y.-C. Liang, A discrete particle swarm optimization algorithm for single machine total earliness and tardiness problem with a common due date. *In: Proc. of the World Congress on Evolutionary Computation* (CEC'06, Vancouver, Canada 2006) 3281–3288.
30. T.C.E. Cheng and H.G. Kahlbacher, A proof for the longest/job/first policy in one/machine scheduling, *Naval Research Logistics* 38 (1991) 715-720.
31. T.-On Ting, M.V.C. Rao, C.K. Loo and S.S. Ngu, Solving Unit Commitment Problem Using Hybrid Particle Swarm Optimization, *Journal of Heuristics* 9 (2003) 507–520.
32. M. Clerc, Discrete particle swarm optimization, illustrated by the traveling salesman problem, *New Optimization Techniques in Engineering*, (Heidelberg, Germany, Springer, 2004) 219-239.
33. Salman, I. Ahmad, S. Al-Madani, Particle swarm optimization for task assignment problem, *Microprocessors and Microsystems* 26 (2003) 363-371.
34. G.C. Onwubolu and M. Clerc, Optimal operational path for automated drilling operations by a new heuristic approach using particle swarm optimization, *Int. Journal of Production Research* 42 (2004) 473-491.
35. S. Liu, J. Tang and J. Song, Order-planning model and algorithm for manufacturing steel sheets, *Int. Journal of Production Economics* 100 (2006) 30-43.
36. A.W. Mohemmed, N. C. Sahoo, T. K. Geok, Solving shortest path problem using particle swarm optimization, *Applied Soft Computing* 8 (2008) 1643-1653.
37. K. Rameshkumar, R.K. Suresh, and K.M. Mohanasundaram, Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan, *in Proc. ICNC 2005*, L. Wang,

K. Chen, and Y.S. Ong (Eds.), Springer-Verlag, Berlin 2005) 572-581.
38. C.–J. Liao, C.-T. Tseng, P. Luarn, A discrete version of particle swarm optimization for flowshop scheduling problems, *Computers & Operations Research* 34 (2007) 3099-3111.
39. Z. Lian, X. Gu and B. Jiao, A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan, *Chaos, Solitons & Fractals* 5 (2008) 851-861.
40. C.-T. Tseng and C.-J. Liao, A discrete particle swarm optimization for lot-streaming flowshop scheduling problem, *European Journal of Operational Research*, 191 (2008) 360-373.
41. Zhang, J. Sun, X. Zhu and Q. Yang, An improved particle swarm optimization algorithm for flowshop scheduling problem, *Information Processing Letters* 108 (2008) 204-209.
42. M.F. Tasgetiren, Y.-C. Liang, M. Sevkli and G. Gencyilmaz, Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem, *Int. Journal of Production Research* 44 (2006) 4737-4754.
43. Allahverdi and F.S. Al-Anzi, Evolutionary heuristics and an algorithm for the two-stage assembly scheduling problem to minimize makespan with setup times, *Int. Journal of Production Research* 44 (2006) 4713-4735.
44. Lei, A Pareto archive particle swarm optimization for multi-objective job shop scheduling, *Computers & Industrial Engineering* 54 (2008) 960-971.
45. T.-L. Lin, S.-J. Horng, T.-W. Kao, Y.-H. Chen, R.-S. Run, R.-J. Chen, J.-L. Lai and I-H. Kuo, An efficient job-shop scheduling algorithm based on particle swarm optimization, *Expert Systems with Applications* 37 (2010) 2629-2636.
46. J. Kennedy, The behavior of particles, In: Porto V.W., Saravanan N, Waagen D. and Eiben A.E. (eds) *Evolutionary Programming VII* (Springer 1998) 581–590.
47. K.E. Parsopoulos and M.N. Vrahatis, Recent approaches to global optimization problems through particle swarm optimization, *Natural Computing* 1 (2002) 235-306.
48. Carlisle and G. Dozier, An Off-The-Shelf PSO, *in Proc. of the Particle Swarm Optimization Workshop* 2001, pp. 1–6
49. Z. Michalewicz and D.B Fogel, *How to solve it: Modern heuristics*, (Springer-Verlag, Berlin 2000).
50. A.E. Eiben and J.E. Smith, *Introduction to evolutionary computing*, (Springer, Berlin 2003).
51. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, Equation of state calculations for fast computing machines, *Journal of Chemical Physics* 21 (1953) 1087-1092.

Co-published by Atlantis Press and Taylor & Francis