International Journal of Production Research

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tprs20

Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem

M. Fatih Tasgetiren [a], Yun-Chia Liang [b], Mehmet Sevkli [c] & Gunes Gencyilmaz [d]

[a] Department of Management, Fatih University, 34500 Buyukcekmece, Istanbul, Turkey

[b] Department of Industrial Engineering and Management, Yuan Ze University, No. 135 Yuan-Tung Road, Chung-Li, Taoyuan County, Taiwan 320, ROC

[c] Department of Industrial Engineering, Fatih University, 34500 Buyukcekmece, Istanbul, Turkey

[d] Department of Management, Istanbul Kultur University, E5 Karayolu Uzeri, Sirinevler, Istanbul, Turkey

Published online: 22 Feb 2007.

To cite this article: M. Fatih Tasgetiren, Yun-Chia Liang, Mehmet Sevkli & Gunes Gencyilmaz (2006) Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem, International Journal of Production Research, 44:22, 4737-4754, DOI: 10.1080/00207540600620849

To link to this article: http://dx.doi.org/10.1080/00207540600620849

International Journal of Production Research,
Vol. 44, No. 22, 15 November 2006, 4737–4754

Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem

M. FATIH TASGETIREN*†, YUN-CHIA LIANG‡,
MEHMET SEVKLI§ and GUNES GENCYILMAZ¶

†Department of Management, Fatih University, 34500 Buyukcekmece, Istanbul, Turkey
‡Department of Industrial Engineering and Management, Yuan Ze University,
No. 135 Yuan-Tung Road, Chung-Li, Taoyuan County, Taiwan 320, ROC
§Department of Industrial Engineering, Fatih University,
34500 Buyukcekmece, Istanbul, Turkey
¶Department of Management, Istanbul Kultur University,
E5 Karayolu Uzeri, Sirinevler, Istanbul, Turkey

(*Revision received February 2006*)

In this paper we present two recent metaheuristics, particle swarm optimization and differential evolution algorithms, to solve the single machine total weighted tardiness problem, which is a typical discrete combinatorial optimization problem. Most of the literature on both algorithms is concerned with continuous optimization problems, while a few deal with discrete combinatorial optimization problems. A heuristic rule, the smallest position value (SPV) rule, borrowed from the random key representation in genetic algorithms, is developed to enable the continuous particle swarm optimization and differential evolution algorithms to be applied to all permutation types of discrete combinatorial optimization problems. The performance of these two recent population based algorithms is evaluated on widely used benchmarks from the OR library. The computational results show that both algorithms show promise in solving permutation problems. In addition, a simple but very efficient local search method based on the variable neighbourhood search (VNS) is embedded in both algorithms to improve the solution quality and the computational efficiency. Ultimately, all the best known or optimal solutions of instances are found by the VNS version of both algorithms.

Keywords: Particle swarm optimization; Differential evolution; Total weighted tardiness; Single machine scheduling problem; Evolutionary algorithms

1. Introduction

Scheduling is a form of decision-making that plays a crucial role in manufacturing as well as in service industries. In the modern competitive environment, effective scheduling has become a necessity for survival in the marketplace. Thus, meeting due dates and avoiding delay penalties are the most typical goals of scheduling. The costs of tardy deliveries, such as a company's goodwill, future sales losses, and

*Corresponding author. Email: ftasgetiren@fatih.edu.tr

International Journal of Production Research
ISSN 0020–7543 print/ISSN 1366–588X online © 2006 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/00207540600620849

rush shipping costs, vary significantly over customers and over orders, and this implied 'strategic weight' should be reflected in job priority. Therefore, minimizing total weighted tardiness for the jobs awaiting completion is not only a measure of academic interest but also useful and important in practice.

McNaughton (1959) was the first to present the scheduling problem in the form where the objective is to minimize total penalty cost. He proved that an optimal solution exists in which no job is split, so that only permutation schedules of the n jobs must be considered. Therefore, the single machine total weighted tardiness (SMTWT) problem can be stated as follows. Each of n jobs ($j = 1, \ldots, n$) in a queue is to be processed without pre-emption on a single machine that can handle no more than one job at a time. The processing and setup requirements of any job are independent of its position in the permutation. The release time of all jobs is zero. Thus, job j ($j = 1, \ldots, n$) becomes available for processing at time zero, requires an uninterrupted positive processing time p_j, which includes setup and knock-down times on the machine, has a positive weight w_j, and has a due date d_j by which it should ideally be finished. For a given processing order of the jobs, the tardiness $T_j = \max\{0, C_j - d_j\}$ of job j ($j = 1, \ldots, n$) can be computed. Then, the SMTWT problem is to find a processing order of the jobs that minimizes the sum of total weighted tardiness over all jobs, i.e. $\sum_{j=1}^{n} w_j T_j$. Moreover, this problem permits direct comparison of sequencing procedures without regard to either the random effects of job arrivals or to the due date setting methodology. For the computational complexity of the SMTWT problem, Lawler (1964) and Lenstra *et al.* (1977) proved that it is NP-hard.

Particle Swarm Optimization (PSO) and Differential Evolution (DE) are two of the latest metaheuristic methods. Although the applications of both PSO and DE on combinatorial optimization problems are still limited, both algorithms have similar advantages such as simple concept, immediately accessible for practical applications, simple structure, ease of use, speed of obtaining solutions, and robustness. However, the continuous nature of both algorithms prohibits the direct application to combinatorial optimization problems. To remedy this drawback, Tasgetiren *et al.* (2004a,b) present the smallest position value (SPV) rule, borrowed from the *random key representation* of Bean (1994), for the PSO algorithm to convert a continuous position vector into a discrete job permutation. Following the successful applications above, this paper aims at employing both PSO and DE to solve the SMTWT problem. This paper is organized as follows. Section 2 gives a brief review of the PSO and DE literature, and the SMTWT problem. In section 3, the methodology of the PSO algorithm is discussed, and the DE algorithm is proposed in section 4. An enhanced local search mechanism embedded in both the PSO and DE algorithms is introduced in section 5. Computational results of test problems are shown in section 6. Finally, section 7 summarizes the concluding remarks.

2. Literature review

2.1 *PSO and DE literature*

PSO is based on the metaphor of social interaction and communication such as bird flocking and fish schooling. PSO is distinctly different from other evolutionary-type

methods in that it does not use the filtering operation (such as crossover and/or mutation) and the members of the entire population are maintained through the search procedure so that information is socially shared among individuals to direct the search towards the best position in the search space. In a PSO algorithm, each member is called a *particle*, and each particle moves around in the multi-dimensional search space with a velocity that is constantly updated by the particle's own experience and the experience of the particle's neighbours or the experience of the whole swarm.

Like the real-coded genetic algorithm (GA), candidate solutions in DE are represented as individuals based on floating-point numbers. In the DE algorithm, the target population is perturbed with a mutant factor, and the crossover operator is then introduced to combine the mutated population with the target population so as to generate a trial population. Then the selection operator is applied to compare the fitness function value of both competing populations, namely, the target and trial population. Ultimately, better individuals become members of the population for the next generation. This process is repeated until convergence occurs.

PSO and DE were both first introduced to optimize various continuous nonlinear functions by Eberhart and Kennedy (1995) and Storn and Price (1995, 1997), respectively. PSO has been successfully applied to a wide range of applications, and comprehensive surveys of PSO can be found in Kennedy *et al.* (2001) and of DE in Lampinen (2001), and Onwubolu and Babu (2004).

2.2 *SMTWT problem literature*

Since the SMTWT problem was first discussed in the late 1950s, it has attracted much research. This research can be categorized into three main groups: exact solution methods, dispatching rules, and heuristic approaches. Exact solution methods consist of dynamic programming and branch-and-bound algorithms. Both use dominance rules to restrict the search for the optimal solution and engender computational practicality. Rinnooy Kan *et al.* (1975) extended Emmons' (1969) rules to the SMTWT problem. Dynamic programming formulations of the SMTWT problem have been presented by Held and Karp (1962), Lawler (1964), Baker and Schrage (1978), and Schrage and Baker (1978). Abdul-Razaq *et al.* (1990) developed general recursion equations for all well-known dynamic programming algorithms. None of these DP algorithms can consistently solve problems with more than 30 jobs within an acceptable computation time. Branch-and-bound approaches were proposed by Shwimer (1972), Rinnooy Kan *et al.* (1975), Potts and Van Wassenhove (1985), Abdul-Razaq *et al.* (1990), and Babu *et al.* (2004). Shwimer and Rinnooy's algorithms required very large computation times because weak lower bounds were used. Potts's algorithm solved problems up to 50 jobs. Babu's work integrated a lower bound based on a Lagrangian decomposition with their branch-and-bound algorithm.

All enumerative algorithms for the SMTWT problem require substantial demands on computation time and/or memory usage, especially when the number of jobs is more than 50. Therefore, a variety of dispatching rules have been proposed to solve problems quickly. Popular rules for the SMTWT problem include EDD

(Earliest Due Date), WSPT (Weighted Shortest Processing Time), WMR (Weighted Montagne's Ratio), WCR (Weighted Critical Ratio), MCOVERT (Modified Cost OVER Time), WCOVERT (Weighted Cost OVER Time), API (Apparent Priority Index), AU (Apparent Urgency), and COVERT-AU. Potts and Van Wassenhove (1991) compared WSPT, EDD, MCOVERT, and AU, and found that AU performs best and WSPT is greatly inferior. Alidaee and Ramakrishnan (1996) provided computational tests of the COVERT-AU class of dispatching rules for problems of up to 200 jobs. Dispatching rules offer a quick sequencing method, but have poor worst-case performance.

Since there is no individual best dispatching rule for all problem environments, researchers have paid more attention to meta-heuristic methods, such as simulated annealing (SA), GA, tabu search (TS), and ant colony optimization (ACO), which offer a compromise between computational practicality and near-optimal sequencing. Matsuo *et al.* (1989) used a controlled search SA (CSSA) to solve the SMTWT problem. Four dispatching rules, EDD, WSPT, MCOVERT, and AU, were used to generate initial solutions and the SA with AU was found to be superior. Crauwels *et al.* (1998) compared several heuristic methods, including descent, threshold accepting, TS, SA, and GA, with two different solution representations—binary encoding and natural permutation. The TS algorithm dominated other methods, although the binary encoded GA generated solutions with good quality. Using ACO for the SMTWT problem has been the subject of two prior papers by den Besten *et al.* (2000) and Merkle and Middendorf (2003). Den Besten *et al.* applied an ant colony system (ACS) to a set of test problems of size up to 100 jobs from the literature with excellent results. Three different problem-specific heuristics and five types of local search were tested. Merkle and Middendorf used pheromone summation evaluation for the transition probability. The authors tested their algorithm for both weighted and unweighted problems, and the results were worse than den Besten's. Den Besten *et al.* (2001) also presented an iterated local search (ILS) algorithm to solve the SMTWT problem, and their algorithm was able to reach all optimum or best solutions of a set of well-known benchmark problems.

Congram *et al.* (2002) proposed an iterated dynasearch algorithm for the SMTWT problem. The authors developed an approach which allows a series of moves to be performed at each generation while traditional local search makes a single move only. The AU dispatching rule was used to generate the initial solution. Problems sizes of up to 100 jobs were tested and this iterated algorithm outperformed the best algorithm, tabu search, in Crauwels *et al.* (1998). Grosso *et al.* (2004), based on Congram's work, developed an enhanced dynasearch neighbourhood using the generalized pairwise interchange (GPI) method. Maheswaran and Ponnambalam (2003) presented a backward forward heuristic to solve the SMTWT problem. The authors used the benchmarks in the OR library to investigate the influence of Relative Due Date (RDD) and Tardiness Factor (TF) on problem instances. The authors in another work (Maheswaran and Ponnambalam 2005) proposed an intensive search evolutionary algorithm for the SMTWT problem. An ordered crossover and a random swap mutation operator were applied. The test results on the OR library benchmarks improved the performance of their previous research.

3. Particle swarm optimization algorithm

The PSO algorithm for the SMTWT problem begins with initializing parameters and generating the initial population randomly. Then the SPV rule applies to each particle to find its permutation so that the fitness value, the total weighted tardiness, of the particle can be evaluated. After evaluation, the PSO algorithm repeats the following steps iteratively.

Initially, each individual with its position, velocity, and fitness value is assigned to its personal best (i.e. the best value of each individual so far). The best individual in the whole swarm, with its position, velocity, and fitness value, is, on the other hand, assigned to the global best (i.e. the best particle in the whole swarm). Then each particle updates its velocity based on the experiences of the personal best and the global best in order to update the position of each particle with the velocity currently updated. The corresponding permutation is determined through the SPV rule so that evaluation is again performed to compute the fitness of the particles in the swarm. In addition, a local search may apply to a certain group of particles in the swarm to enhance the exploitation of the search space. This process is terminated with a predetermined stopping criterion. We follow the *gbest* model of Eberhart and Kennedy (1995) with the inclusion of the SPV rule in the algorithm. The pseudo-code of the PSO algorithm for the SMTWT problem is given in figure 1.

The basic elements of the PSO algorithm are summarized as follows.

Particle. X_i^t denotes the ith particle in the swarm at iteration t and is defined as $X_i^t = [x_{i1}^t, x_{i2}^t, \ldots, x_{in}^t]$, where x_{ij}^t is the position value of the ith particle with respect to the jth dimension ($j = 1, 2, \ldots, n$).

Population. X^t is the set of NP particles in the swarm at iteration t, i.e. $X^t = [X_1^t, X_2^t, \ldots, X_{NP}^t]$, where NP denotes the population size.

Permutation. We introduce a new variable π_i^t, which is a permutation of jobs implied by the particle X_i^t. It can be described as $\pi_i^t = [\pi_{i1}^t, \pi_{i2}^t, \ldots, \pi_{in}^t]$, where π_{ij}^t is the assignment of job j of particle i in the permutation at iteration t.

```
Initialize parameters
Initialize population
Find permutation
Evaluate fitness
Do {
        Find the personal best
        Find the global best
        Update velocity
        Update position
        Find permutation
        Evaluate fitness
        Apply local search
} While (Termination)
```

Figure 1. PSO algorithm with local search for the SMTWT problem.

Particle velocity. V_i^t is the velocity of particle i at iteration t and is defined as $V_i^t = [v_{i1}^t, v_{i2}^t, \ldots, v_{in}^t]$, where v_{ij}^t is the velocity of particle i at iteration t with respect to the jth dimension ($j = 1, 2, \ldots, n$).

Inertia weight. w^t is a parameter to control the impact of previous velocities on the current velocity.

Fitness function. In a minimization problem, the objective function is $f_i(\pi_i^t \leftarrow X_i^t)$, where π_i^t is the corresponding permutation of particle X_i^t.

Personal best. P_i^t represents the best position of particle i with the best fitness until iteration t, and is called the *personal best*. For each particle in the swarm, P_i^t can be determined and updated at each iteration t. In a minimization problem with objective function $f(\pi_i^t \leftarrow X_i^t)$, the personal best P_i^t of the ith particle is obtained such that $f(\pi_i^t \leftarrow P_i^t) \leq f(\pi_i^{t-1} \leftarrow P_i^{t-1})$ for $i = 1, 2, \ldots, NP$. To simplify, we denote the fitness function of the personal best as $f_i^{\text{pb}} = f(\pi_i^t \leftarrow P_i^t)$. For each particle, the personal best is defined as $P_i^t = [p_{i1}^t, p_{i2}^t, \ldots, p_{in}^t]$, where p_{ij}^t denotes the position value of the ith personal best with respect to the jth dimension ($j = 1, 2, \ldots, n$).

Global best. G^t denotes the best position of the globally best particle achieved so far in the whole swarm. Therefore, the global best can be obtained such that $f(\pi^t \leftarrow G^t) \leq f(\pi_i^t \leftarrow P_i^t)$ for $i = 1, 2, \ldots, NP$. To simplify, we denote the fitness function of the global best as $f^{\text{gb}} = f(\pi^t \leftarrow G^t)$. The global best is then defined as $G^t = [g_1^t, g_2^t, \ldots, g_n^t]$, where g_j^t is the position value of the global best with respect to the jth dimension ($j = 1, 2, \ldots, n$).

3.1 *Solution representation*

One of the most important issues when designing the PSO and DE algorithms lies in its solution representation. In order to construct a direct relationship between the problem domain and the PSO particles or DE individuals for the SMTWT problem, each dimension represents a typical job. In addition, the particle or individual $X_i^t = [x_{i1}^t, x_{i2}^t, \ldots, x_{in}^t]$ corresponds to the continuous position values for n jobs in the SMTWT problem. The particle or individual itself does not present a permutation. Instead, we use the SPV rule to determine the permutation implied by the particle or individual X_i^t. This representation is unique in terms of finding new solutions since the particle or individual is updated at each iteration t, thus resulting in different permutations at each iteration t. Table 1 illustrates the solution representation of particle or individual X_i^t for the PSO and DE algorithms. According to the SPV rule, the smallest position value is $x_{i5}^t = -1.20$, so the dimension $j = 5$ is assigned to be the first job $\pi_{i1}^t = 5$ in the permutation π_i^t; the second smallest position value is $x_{i2}^t = -0.99$, so the dimension $j = 2$ is assigned to be the second job $\pi_{i2}^t = 2$ in the permutation π_i^t, and so on. In other words, a random key is used to construct the permutation π_i^t.

Table 1. Solution representation of particle or individual X_i^t.

Dimension, j	1	*2*	3	4	**5**
x_{ij}^t	1.80	*−0.99*	3.01	−0.72	**−1.20**
Job, π_{ij}^t	**5**	*2*	4	1	3

3.2 *Initial population*

A population of particles or individuals is constructed randomly for the PSO and DE algorithms. The continuous values of positions or individuals are established randomly. The following formula is used to construct the initial continuous position values uniformly:

$$x_{ij}^0 = x_{\min} + (x_{\max} - x_{\min}) * r_1, \tag{1}$$

where $x_{\min} = -1.0$, $x_{\max} = 1.0$ and r_1 is a uniform random number between 0 and 1. Initial velocities for the PSO particles are also generated by a similar formula as follows:

$$v_{ij}^0 = v_{\min} + (v_{\max} - v_{\min}) * r_2, \tag{2}$$

where, $v_{\min} = -1.0$, $v_{\max} = 1.0$ and r_2 is a uniform random number between 0 and 1.

The population size is ten times the number of dimensions. As the formulation of the SMTWT problem suggests that the objective is to minimize the total weighted tardiness, the fitness function value is the total weighted tardiness for the particle or individual i of the population. That is,

$$f_i^t(\pi_i^t \leftarrow X_i^t) = \sum_{j=1}^{n} w_{ij} T_{ij}. \tag{3}$$

For simplicity, $f_i^t(\pi_i^t \leftarrow X_i^t)$ will be denoted f_i^t. The complete computational procedure of the PSO algorithm for the SMTWT problem can be summarized as follows.

Step 1: Initialization

- Set $t = 0$, $NP =$ ten times the number of dimensions.
- Generate NP particles randomly using equation (1), $\{X_i^0,\ i = 1, 2, \ldots, NP\}$, where $X_i^0 = [x_{i1}^0, x_{i2}^0, \ldots, x_{in}^0]$.
- Generate the initial velocities for particle i randomly as in equation (2), $\{V_i^0, i = 1, 2, \ldots, NP\}$, where $V_i^0 = [v_{i1}^0, v_{i2}^0, \ldots, v_{in}^0]$.
- Apply the SPV rule to find the permutation $\pi_i^0 = [\pi_{i1}^0, \pi_{i2}^0, \ldots, \pi_{in}^0]$ of particle X_i^0 for $i = 1, 2, \ldots, NP$.
- Evaluate each particle i in the swarm using the objective function f_i^0 for $i = 1, 2, \ldots, NP$.
- For each particle i in the swarm, set the personal best to $P_i^0 = X_i^0$, where $P_i^0 = [p_{i1}^0 = x_{i1}^0, p_{i2}^0 = x_{i2}^0, \ldots, p_{in}^0 = x_{in}^0]$, together with its best fitness value, $f_i^{\text{pb}} = f_i^0$ for $i = 1, 2, \ldots, NP$.
- Find the best fitness value among the whole swarm such that $f_l = \min\{f_i^0\}$ for $i = 1, 2, \ldots, NP$ with its corresponding positions X_l^0. Set the global best to $G^0 = X_l^0$ such that $G^0 = [g_1 = x_{l,1}, g_2 = x_{l,2}, \ldots, g_n = x_{l,n}]$ with its fitness value $f^{\text{gb}} = f_l$.

Step 2: Update iteration counter

- $t = t + 1$.

Step 3: Update inertia weight

- $w^t = w^{t-1} * \beta$, (4)

  where β is the decrement factor.

Step 4: Update velocity

- $v_{ij}^t = w^{t-1} v_{ij}^{t-1} + c_1 r_1 (p_{ij}^{t-1} - x_{ij}^{t-1}) + c_2 r_2 (g_j^{t-1} - x_{ij}^{t-1}),$ (5)

  where c_1 and c_2 are the social and cognitive parameters and r_1 and r_2 are uniform random numbers between (0, 1).

Step 5: Update position

- $x_{ij}^t = x_{ij}^{t-1} + v_{ij}^t.$ (6)

Step 6: Find permutation

- Apply the SPV rule to find the permutation $\pi_i^t = [\pi_{i1}^t, \pi_{i2}^t, \ldots, \pi_{in}^t]$ for $i = 1, 2, \ldots, NP$.

Step 7: Update the personal best

- Each particle is evaluated by using the permutation to see if the personal best will improve. That is, if $f_i^t < f_i^{\text{pb}}$ for $i = 1, 2, \ldots, NP$ then the personal best is updated as $P_i^t = X_i^t$ and $f_i^{\text{pb}} = f_i^t$ for $i = 1, 2, \ldots, NP$.

Step 8: Update the global best

- Find the minimum value of the personal best. That is, $f_l^t = \min\{f_i^{\text{pb}}\}$, $i = 1, 2, \ldots, NP$; $l \in \{i; i = 1, 2, \ldots, NP\}$.
- If $f_l^t < f^{\text{gb}}$, then the global best is updated as $G^t = X_l^t$ and $f^{\text{gb}} = f_l^t$.

Step 9: Stopping criterion

- If the number of iterations exceeds the maximum number of iterations, then stop; otherwise go to step 2.

4. Differential evolution algorithm

Solution representation and the initial population in the DE algorithm are the same as those in the PSO algorithm, but omitting the velocity vector in the representation. Currently, there are several variants of the DE algorithm. We follow the *DE/rand/1/bin* scheme of Storn and Price (1995) with the inclusion of the SPV rule in the algorithm. The pseudo-code of the DE algorithm for the SMTWT problem is given in figure 2.

The basic elements of the DE algorithm are summarized as follows.

Target individual. X_i^t denotes the ith individual in the population at generation t and is defined as $X_i^t = [x_{i1}^t, x_{i2}^t, \ldots, x_{in}^t]$, where x_{ij}^t is the parameter value of the ith individual with respect to the jth dimension ($j = 1, 2, \ldots, n$).

Mutant individual. V_i^t denotes the ith individual in the population at generation t and is defined as $V_i^t = [v_{i1}^t, v_{i2}^t, \ldots, v_{in}^t]$, where v_{ij}^t is the parameter value of the ith individual with respect to the jth dimension ($j = 1, 2, \ldots, n$).

Trial individual. U_i^t denotes the ith individual in the population at generation t and is defined as $U_i^t = [u_{i1}^t, u_{i2}^t, \ldots, u_{in}^t]$, where u_{ij}^t is the parameter value of the ith individual with respect to the jth dimension ($j = 1, 2, \ldots, n$).

Initialize parameters
Initialize target population
Find permutation
Evaluate fitness of the target population
Do {
    Obtain the mutant population
    Obtain the trial population
    Find permutation
    Evaluate fitness of the trial population
    Do Selection
    Apply local search
}While (Not Termination)

Figure 2. DE algorithm with local search for the SMTWT problem.

Target population. X^t is the set of NP individuals in the target population at generation t, i.e. $X^t = [X_1^t, X_2^t, \ldots, X_{NP}^t]$.

Mutant population. V^t is the set of NP individuals in the mutant population at generation t, i.e. $V^t = [V_1^t, V_2^t, \ldots, V_{NP}^t]$.

Trial population. U^t is the set of NP individuals in the trial population at generation t, i.e. $U^t = [U_1^t, U_2^t, \ldots, U_{NP}^t]$.

Permutation. We introduce a new variable π_i^t, which is a permutation of jobs implied by individual X_i^t. It can be described as $\pi_i^t = [\pi_{i1}^t, \pi_{i2}^t, \ldots, \pi_{in}^t]$, where π_{ij}^t is the assignment of job j of individual i in the permutation at generation t.

Mutant constant. $F \in (0, 2)$ is a real number constant which affects the differential variation between two individuals.

Crossover constant. $CR \in (0, 1)$ is a real number constant which affects the diversity of the population for the next generation.

Fitness function. In a minimization problem, the objective function is $f_i(\pi_i^t \leftarrow X_i^t)$, where π_i^t is the corresponding permutation of individual X_i^t.

The complete computational procedure of the DE algorithm for the SMTWT problem can be summarized as follows.

Step 1: Initialization

- Set $t = 0$, $NP =$ ten times the number of dimensions.
- Generate NP individuals randomly as in equation (1), $\{X_i^0, i = 1, 2, \ldots, NP\}$, where $X_i^0 = [x_{i1}^0, x_{i2}^0, \ldots, x_{in}^0]$.
- Apply the SPV rule to find the permutation $\pi_i^0 = [\pi_{i1}^0, \pi_{i2}^0, \ldots, \pi_{in}^0]$ of individual X_i^0 for $i = 1, 2, \ldots, NP$.
- Evaluate each individual i in the population using the objective function $f_i^0(\pi_i^0 \leftarrow X_i^0)$ for $i = 1, 2, \ldots, NP$.

Step 2: Update generation counter

- $t = t + 1$.

Step 3: Generate mutant population

- For each target individual, X_i^t, $i = 1, 2, \ldots, NP$, at generation t, a mutant individual, $V_i^t = [v_{i1}^t, v_{i2}^t, \ldots, v_{in}^t]$, is determined such that

$$V_i^t = X_{a_i}^{t-1} + F(X_{b_i}^{t-1} - X_{c_i}^{t-1}), \tag{7}$$

where a_i, b_i, and c_i are three randomly chosen individuals from the population such that $a_i \neq b_i \neq c_i$.

Step 4: Generate trial population

- Following the mutation phase, the crossover (recombination) operator is applied to obtain the trial population. For each mutant individual, $V_i^t = [v_{i1}^t, v_{i2}^t, \ldots, v_{in}^t]$, an integer random number between 1 and n, i.e. $D_i \in (1, 2, \ldots, n)$, is chosen, and a trial individual, $U_i^t = [u_{i1}^t, u_{i2}^t, \ldots, u_{in}^t]$, is generated such that

$$u_{ij}^t = \begin{cases} v_{ij}^t, & \text{if } r_{ij}^t \leq CR \text{ or } j = D_i, \\ x_{ij}^{t-1}, & \text{otherwise,} \end{cases} \tag{8}$$

where the index D refers to a randomly chosen dimension ($j = 1, 2, \ldots, n$), which is used to ensure that at least one parameter of each trial individual U_i^t differs from its counterpart in the previous generation U_i^{t-1}, CR is a user-defined crossover constant in the range $(0, 1)$, and r_{ij}^t is a uniform random number between 0 and 1. In other words, the trial individual is made up of some parameters of the mutant individual, or at least one of the parameters randomly selected, and some other parameters of the target individual.

Step 5: Find permutation

- Apply the SPV rule to find the permutation $\pi_i^t = [\pi_{i1}^t, \pi_{i2}^t, \ldots, \pi_{in}^t]$ for $i = 1, 2, \ldots, NP$.

Step 6: Evaluate trial population

- Evaluate the trial population using the objective function $f_i^t(\pi_i^t \leftarrow U_i^t)$ for $i = 1, 2, \ldots, NP$.

Step 7: Selection

- To decide whether or not the trial individual U_i^t should be a member of the target population for the next generation, it is compared with its counterpart target individual X_i^{t-1} at the previous generation. The selection is based on the survival of fitness among the trial population and target population such that

$$X_i^t = \begin{cases} U_i^t, & \text{if } f(\pi_i^t \leftarrow U_i^t) \leq f(\pi_i^{t-1} \leftarrow X_i^{t-1}), \\ X_i^{t-1}, & \text{otherwise.} \end{cases} \tag{9}$$

Step 8: Stopping criterion

- If the number of generations exceeds the maximum number of generations, then stop; otherwise go to step 2.

5. Neighbourhood of PSO and DE algorithms

In both algorithms, local search is applied to the permutation directly. However, it violates the SPV rule and needs a repair algorithm. This approach is illustrated in tables 2 and 3, where job $\pi^t_{i2} = 2$ and job $\pi^t_{i4} = 1$ are interchanged. As seen in table 2, applying a local search to the permutation violates the SPV rule because the permutation itself is a result of the particle's position or individual's parameter values. Once a local search is completed, the particle or individual should be repaired so that the SPV rule is not violated. This is achieved by changing the position or parameter values according to the SPV rule as shown in table 3. In other words, we interchange the position or parameter values of the exchanged jobs in terms of their dimensions. Since jobs $\pi^t_{i2} = 2$ and $\pi^t_{i4} = 1$ are interchanged, their associated position or parameter values $x^t_{i2} = -0.99$ and $x^t_{i1} = 1.80$ are interchanged for dimensions $j = 2$ and $j = 1$ to keep the particle or individual consistent with the SPV rule. The advantage of this method is due to the fact that the repair algorithm is only needed after evaluating all the neighbours in a permutation.

The local search for the SMTWT problem is applied to the permutation π^t of the global best or the best solution so far at each generation t. The performance of the local search algorithm depends on the choice of the neighbourhood structure. Local search in this work is based on the *insert + interchange* variant of the variable neighbourhood search (VNS) method presented in Mladenovic and Hansen (1997). For the SMTWT problem, the following two neighbourhood structures are employed:

- interchange two jobs between the ηth and κth dimensions, $\eta \neq \kappa$ (*interchange*);
- remove the job at the ηth dimension and insert it in the κth dimension $\eta \neq \kappa$ (*insert*).

The pseudo-code of the local search is given in figure 3, where η and κ are random integer numbers between 1 and n. For convenience, $s = \text{insert}(s_0, \eta, \kappa)$ means

Table 2. Local search applied to permutation before repairing.

Dimension, j	1	2	3	4	5
x^t_{ij}	1.80	−0.99	3.01	−0.72	−1.20
Job, π^t_{ij}	5	**2**	4	**1**	3
x^t_{ij}	1.80	−0.99	3.01	−0.72	−1.20
Job, π^t_{ij}	5	**1**	4	**2**	3

Table 3. Local search applied to permutation after repairing.

Dimension, j	**1**	**2**	3	4	5
x^t_{ij}	**1.80**	**−0.99**	3.01	−0.72	−1.20
Job, π^t_{ij}	5	2	4	**1**	3
x^t_{ij}	**−0.99**	**1.80**	3.01	−0.72	−1.20
Job, π^t_{ij}	5	**1**	4	**2**	3

```
outloop=0;
s0= π^t , permutation of global best G^t ;
do{
   u=rnd(1,n); v=rnd(1,n);
  s=insert(s0, u, v)
   inloop=0;
   do{
       kcount=0; max_method=2;
      do{
            u=rnd(1,n); v=rnd(1,n);
        if (kcount=0) then s1=insert(s, u, v)
        if (kcount=1) then s1=interchange(s, u, v)
      if    (f(s1) ≤ f(s))   then{
                              kcount=0;
                              s= s1;}
       else { kcount++;}
       }while (kcount<max_method)
    inloop++;
    }while (inloop<n*(n-1))
    outloop++;
    if    (f(s1) ≤ f(π^t))   then{
                              π^t =s;
              repair( G^t );}
}while (outloop< p_ls * popsize );
```

Figure 3. Pseudo-code of the VNS local search employed.

removing the job from the ηth dimension in the permutation s_0 and inserting it in the κth dimension in the permutation s_0, thus resulting in permutation s. The probability of the local search, p_{ls}, is taken as 0.10. In other words, the local search is applied to the permutation π^t of global best solution G^t at each generation p_{ls} times the population size.

Note that neutral moves are allowed in the VNS local search. For simplicity, PSO and DE with the SPV rule alone will be denoted PSO_{spv} and DE_{spv}, respectively, while PSO and DE with an extra VNS local search will be denoted PSO_{vns} and DE_{vns}, respectively, throughout the paper from now on.

6. Experimental results

The proposed PSO and DE algorithms for the SMTWT problem are coded in C and run on an Intel Pentium IV 2.6 GHz PC with 256 MB memory. The performance of PSO_{spv} and DE_{spv} is first presented and then the performance of PSO_{vns} and DE_{vns} is evaluated. We used the following parameters for the PSO and DE algorithms. The size of the population both in the PSO and DE algorithms is 10 times the number of dimensions. Social and cognitive parameters are taken as $c_1 = c_2 = 2$ consistent with the literature. Initial inertia weight is set to $w^0 = 0.9$ and never decreased below 0.40. Finally, the decrement factor β is taken as 0.975. For the DE algorithm, the mutant factor and crossover rate are taken as $F = 0.4$ and $CR = 0.5$, respectively. For both algorithms, the maximum number of iterations or generations is set to 2000,

where we stop the search when the best known or optimal solution is found. Twenty-five replications are carried out to collect the statistics.

We evaluated the performance of both algorithms using the benchmark set of randomly generated instances, available from the OR Library (URL: http://people.brunel.ac.uk/~mastjjb/jeb/info.html). There are three sets of problems with 40, 50 and 100 jobs, each containing 125 instances. The optimal solutions for the 40 and 50 job instances are known. However, the 100 job instances have not been solved to optimality so far and best known solutions are available. The instances in the OR library were randomly generated as follows. For each job j ($j = 1, \ldots, n$), an integer processing time p_j was generated from the uniform distribution (1, 100) and an integer processing weight w_j was generated from the uniform distribution (1, 10). Using different uniform distributions for generating the due dates generated instance classes of varying hardness. For a given relative range of due dates RDD (RDD = 0.2, 0.4, 0.6, 0.8, 1.0) and a given average tardiness factor TF (TF = 0.2, 0.4, 0.6, 0.8, 1.0), the integer due date d_j for each job j was randomly generated from the uniform distribution

$$\left[P \cdot \left(1 - \mathrm{TF} - \frac{\mathrm{RDD}}{2}\right), P \cdot \left(1 - \mathrm{TF} + \frac{\mathrm{RDD}}{2}\right)\right],$$

where $P = \sum_{j=1}^{n} p_j$. Five instances were generated for each of the 25 pairs of values of RDD and TF, yielding 125 instances for both values of $n = 40$, $n = 50$ and $n = 100$.

The solution quality is measured by the percentage of relative increase in the total weighted tardiness (Δ) with respect to the best known or optimal solutions provided in the OR library. To be more specific, Δ_{avg} is computed as follows:

$$\Delta_{\mathrm{avg}} = \sum_{i=1}^{R} \left(\frac{(H_i - U_i) * 100}{U_i}\right) \Big/ R, \tag{10}$$

where H_i denotes the value of the total weighted tardiness generated by the PSO or DE algorithm, whereas U_i is the value of the best known or optimal solution provided in OR library, and R is the total number of replications, i.e. R is equal to the product of the number of replications for each instance and the total number of instances. That is, with 25 replications, each of 125 problem instances, a total of 3125 runs are conducted for each algorithm for each problem size. In addition, Δ_{avg} denotes the mean percentage relative increase in the total weighted tardiness over R runs, Δ_{std} represents the standard deviation of the percentage of relative increase in the total weighted tardiness, and n_{opt} is the number of best known or optimal solutions found in R runs. For the computational effort required, t_{avg} denotes the average CPU time in seconds over R runs. Finally, the hit ratio, n_{hit}, represents the percentage of optimal or best known solutions found in R runs. First, we report the performance of both algorithms using the SPV rule. Then the impact of the VNS local search alone on the solution quality is analysed to justify its use in both algorithms. Finally, the impact of including the VNS local search in both algorithms on the solution quality and computational expense is investigated.

As shown in table 4, PSO produces a slightly smaller average percentage of the relative increase in total weighted tardiness (Δ_{avg}) than DE for the 40 and 50 job problems, while for the 100 job problem both algorithms produce almost the same deviation from the best known solution. In terms of the standard deviation of the

Table 4. Comparison of PSO_{spv}, DE_{spv} and VNS over R runs.

	PSO_{spv}					DE_{spv}					VNS				
No. of jobs	Δ_{avg}	Δ_{std}	t_{avg}	n_{opt}	n_{hit}	Δ_{avg}	Δ_{std}	t_{avg}	n_{opt}	n_{hit}	Δ_{avg}	Δ_{std}	t_{avg}	n_{opt}	n_{hit}
40	1.31	8.09	6.02	1958	0.63	1.61	10.01	3.04	2360	0.76	0.21	1.33	0.18	2955	0.95
50	1.10	4.56	13.72	1495	0.48	1.24	6.16	8.53	1913	0.61	0.20	2.18	0.40	2833	0.91
100	2.35	11.85	85.37	810	0.26	2.31	13.72	66.63	739	0.24	0.11	1.63	5.72	2804	0.90

relative percentage increase in total weighted tardiness, the PSO algorithm is more robust than the DE algorithm. However, the DE algorithm is much faster than the PSO algorithm. In addition, the DE algorithm is able to find more best known solutions or optimal solutions than the PSO algorithm, since the hit ratio, n_{hit}, of the 40 and 50 job problems for the DE algorithm is 0.76 and 0.61, respectively. However, for the difficult 100 job problem, the hit ratio of PSO is slightly better than that of DE. From the results we can conclude that both algorithms are able to find solutions with deviations from the optimal or best known solutions ranging from 1.10 to 2.35%. Another important observation is that the DE algorithm is computationally less expensive than the PSO algorithm, which is a desirable feature when designing an optimization method. In addition, we wanted to see the impact of the VNS local search alone. For each problem size, we generated 400, 500 and 1000 random solutions. The best solution was then determined from these solutions for each problem size, to be used as the seed permutation for the VNS local search, which was run for 25 replications for each problem instance. The VNS results are also illustrated in table 4. It is clear that the pure VNS local search is superior to both the PSO_{spv} and DE_{spv} algorithms. However, the VNS search alone still cannot find all the optima in all runs; that is, our challenge is to find all the best known or optimal solutions in each single run for all problem instances for all problem sizes, i.e. with a hit ratio of 1.00.

Since the VNS itself is a local search method and generates superior results, it is a good candidate to embed with both the PSO and DE algorithms (as explained in section 5), which are global search algorithms, in order to compensate for the drawback of PSO and DE. By doing so, both algorithms, PSO_{vns} and DE_{vns}, are able to find the best known or optimal solution over all runs for all problem instances in different problem sizes. Since the inclusion of the VNS local search produces no relative error, we want to evaluate the performance of both algorithms in terms of CPU time and the average number of iterations in PSO_{vns} or the average number of generations in DE_{vns}, denoted n_{avg}, in which they find the best known or optimal solutions.

As can be seen in table 5, the average CPU times for PSO_{vns} and DE_{vns} are reduced significantly for all problem sizes when compared with the PSO_{SPV} and DE_{SPV} algorithms. The average CPU times for both the PSO_{vns} and DE_{vns} algorithms are not significantly different. However, the standard deviation of the CPU time for the PSO_{vns} algorithm is smaller than that for the DE_{vns} algorithm in the 100 job category. This indicates that, for the larger instance of the 100 job problem, the DE_{vns} algorithm has difficulty in finding the best known solution in a robust way. When compared with the pure VNS search, both the PSO_{vns} and DE_{vns} algorithms outperform VNS alone with a hit ratio of 1.00. Moreover, both

Table 5. Computational results with reduced VNS local search over R runs.

	PSO_{vns}			DE_{vns}		
No. of jobs	t_{avg}	t_{std}	n_{avg}	t_{avg}	t_{std}	n_{avg}
40	0.21	0.14	1.13	0.20	0.11	1.10
50	0.54	0.86	1.29	0.54	0.91	1.32
100	8.52	19.62	1.44	8.70	28.60	1.49

algorithms are able to find all the best known and optimal solutions in each single run within 1.5 generations, on average, for all problem sizes. By hybridizing the VNS search with the PSO and DE algorithms, we have succeeded in reducing the computational expense and improving the solution quality.

7. Conclusions

To the best of our knowledge, this is the first reported application of a differential evolution algorithm to the single machine total weighted tardiness problem. In this paper, we present the SPV rule based on the random key representation of Bean (1994) for the application of the continuous PSO and DE algorithms to all permutation types of combinatorial optimization problems, and hence to sequencing and scheduling problems. We also hope that the PSO and DE algorithms with the random key representation will enrich the PSO and DE literature, and be applied to combinatorial optimization problems in the future. In addition, both the PSO and DE methods are still in the pioneering stage and much research is required to investigate the impact of employing operators such as fitness scaling, mutation, immigration, constriction factor, craziness operator, etc. on the solution quality.

In this study, the performance of the random key representation is first evaluated through the use of well-known benchmarks from the OR library. The algorithms, in general, generated similar results. However, we observed that the DE algorithm is faster than the PSO algorithm. This feature of the DE algorithm may have a significant impact on the solution quality for larger sized problems, since DE is able to make many more function evaluations than PSO within the same CPU time.

An effective local search, so-called VNS, was then introduced. From the computational results, we have shown that all the statistics are improved. It should be noted that the success is due to the VNS local search, since both algorithms need less than 1.5 generations, on average, to find all optimal or best known solutions for all problems. However, the VNS local search alone cannot achieve a hit ratio of 1.00. Another interpretation of this conclusion is that hybridizing the VNS local search in population-based optimization algorithms helps to improve both the solution quality and the computational efficiency.

In summary, the results presented in this article are very encouraging and promising for the application of the PSO and DE algorithms to the single machine total weighted tardiness problem, and hence to other scheduling problems. For future work, this study may be extended to larger instances to evaluate the impact of the SPV rule. No mutation operator or problem-specific information was used in either algorithm reported here. For larger instances, including problem-specific information in the initial population and applying a mutation operator may improve the performance of both algorithms.

References

Abdul-Razaq, T.S., Potts, C.N. and Van Wassenhove, L.N., A survey of algorithms for the single machine total weighted tardiness scheduling problems. *Discr. Appl. Math.*, 1990, **26**, 235–253.

Alidaee, B. and Ramakrishnan, K.R., A computational experiment of COVERT-AU class of rules for single machine tardiness scheduling problem. *Comput. Ind. Engng*, 1996, **30**, 201–209.

Babu, P., Peridy, L. and Pinson, E., A branch and bound algorithm to minimize total weighted tardiness on a single processor. *Ann. Oper. Res.*, 2004, **129**, 33–46.

Baker, K.R. and Schrage, L.E., Finding an optimal permutation by dynamic programming: an extension to precedence-related tasks. *Oper. Res.*, 1978, **26**, 111–120.

Bean, J.C., Genetic algorithm and random keys for sequencing and optimization. *ORSA J. Comput.*, 1994, **6**, 154–160.

Congram, R.K., Potts, C.N. and Van de Velde, S.L., An iterated dynasearch algorithm for the single machine total weighted tardiness scheduling problem. *INFORMS J. Comput.*, 2002, **14**, 52–67.

Crauwels, H.A., Potts, C.N. and Van Wassenhove, L.N., Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS J. Comput.*, 1998, **10**, 341–350.

den Besten, M.L., Stützle, T. and Dorigo, M., Ant colony optimization for the total weighted tardiness problem, in *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature (PPSN-VI)*, LNCS 1917, 2000, pp. 611–620.

den Besten, M.L., Stützle, T. and Dorigo, M., Design of iterated local search algorithm: an example application to the single machine total weighted tardiness problem. In *Applications of Evolutionary Computing*, edited by E.J.W. Boer, S. Cagnoni, J. Gottlieb, E. Hart, P.L. Lanzi, G.R. Raidl, R.E. Smith and H. Tijink, LNCS 2037, pp. 441–451, 2001.

Eberhart, R.C. and Kennedy, J., A new optimizer using particle swarm theory, in *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43.

Emmons, H., One-machine sequencing to minimize certain functions of job tardiness. *Oper. Res.*, 1969, **17**, 701–715.

Grosso, A., Della Groce, F. and Tadei, R., An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Oper. Res. Lett.*, 2004, **32**, 68–72.

Held, M. and Karp, R.M., A dynamic programming approach to sequencing problems. *J. Soc. Ind. Appl. Math.*, 1962, **10**, 196–210.

Kennedy, J., Eberhart, R.C. and Shi, Y., *Swarm Intelligence*, 2001 (Morgan Kaufmann: San Mateo, CA).

Lampinen, J., A bibliography of differential evolution algorithms. Technical Report, Laboratory of Information Processing, Department of Information Technology, Lappeenranta University of Technology, 2001.

Lawler, E.L., On scheduling problems with deferral costs. *Mgmt Sci.*, 1964, **11**, 280–288.

Lenstra, J.K., Rinnooy Kan, A.H.G. and Brucker, P., Complexity of machine scheduling problems. *Ann. Discr. Math.*, 1977, **1**, 343–362.

Maheswaran, R. and Ponnambalam, S.G., An investigation on single machine total weighted tardiness scheduling problems. *Int. J. Adv. Mnfg Technol.*, 2003, **22**, 243–248.

Maheswaran, R. and Ponnambalam, S.G., An intensive search evolutionary algorithm for single-machine total-weighted-tardiness scheduling problems. *Int. J. Adv. Mnfg Technol.*, 2005, **26**, 1150–1156.

Matsuo, H., Suh, C.J. and Sullivan, R.S., A controlled search simulated annealing method for the single machine weighted tardiness problem. *Ann. Oper. Res.*, 1989, **21**, 85–108.

McNaughton, R., Scheduling with deadlines and loss functions. *Mgmt Sci.*, 1959, **6**, 1–12.

Merkle, D. and Middendorf, M., An ant algorithm with global pheromone evaluation for scheduling a single machine. *Appl. Intell.*, 2003, **18**, 105–111.

Mladenovic, N. and Hansen, P., Variable neighborhood search. *Comput. Oper. Res.*, 1997, **24**, 1097–1100.

Onwubolu, G.C. and Babu, B.V., *New Optimization Techniques in Engineering*, 2004 (Springer: New York).

Potts, C.N. and Van Wassenhove, L.N., A branch and bound algorithm for the total weighted tardiness problem. *Oper. Res.*, 1985, **33**, 363–377.

Potts, C.N. and Van Wassenhove, L.N., Single machine tardiness sequencing heuristics. *IIE Trans*., 1991, **23**, 346–354.

Rinnooy Kan, A.H.G., Lageweg, B.J. and Lenstra, J.K., Minimizing total costs in one-machine scheduling. *Oper. Res*., 1975, **23**, 908–927.

Schrage, L.E. and Baker, K.R., Dynamic programming solution of sequencing problems with precedence constraints. *Oper. Res*., 1978, **26**, 444–449.

Shwimer, J., On the *N*-job, one-machine, permutation-independent scheduling problem with tardiness penalties: a branch-bound solution. *Mgmt Sci*., 1972, **18**, B301–B313.

Storn, R. and Price, K., Differential evolution a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI, 1995.

Storn, R. and Price, K., Differential evolution—A simple and efficient heuristic for global optimization over continuous space. *J. Global Optim*., 1997, **11**, 341–359.

Tasgetiren, M.F., Sevkli, M., Liang, Y.-C. and Gencyilmaz, G., Particle swarm optimization algorithm for single machine total weighted tardiness problem, in *Proceedings of the Congress on Evolutionary Computation (CEC2004)*, 2004a, pp. 1412–1419.

Tasgetiren, M.F., Sevkli, M., Liang, Y.-C. and Gencyilmaz, G., Particle swarm optimization algorithm for permutation flowshop sequencing problem, in *Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS2004)*, LNCS 3172, 2004b, pp. 382–390.