Comput Optim Appl (2012) 51:411–435
DOI 10.1007/s10589-010-9333-7

A GRASP based on DE to solve single machine scheduling problem with SDST

Hanen Akrout · Bassem Jarboui · Patrick Siarry · Abdelwaheb Rebaï

Received: 7 September 2009 / Published online: 26 May 2010

Abstract When handling combinatorial optimization problems, we try to get the optimal arrangement of discrete entities so that the requirements and the constraints are satisfied. These problems become more and more important in various industrial and academic fields. So, over the past years, several techniques have been proposed to solve them. In this paper, we are interested in the single machine scheduling problem with Sequence-Dependent Setup Times, which can be solved through different approaches. We present a hybrid algorithm which combines Greedy Randomized Adaptive Search Procedure and Differential Evolution for tackling this problem. Our algorithm is tested on benchmark instances from the literature. The computational experiments prove the efficiency of this algorithm.

Keywords Scheduling single machine · Greedy randomized adaptive search procedure · Differential evolution · Sequence-dependent setup times · Total tardiness

1 Introduction

Over the last ten years the importance of industrial problems gained a tremendous grow. Many problems have been studied in this field, such as the single machine scheduling problem with Sequence-Dependent Setup Times (SDST). This problem has attracted much attention in production planning. Indeed machine scheduling is

H. Akrout · B. Jarboui · A. Rebaï
GIAD, FSEGS, Route de l'Aéroport km 4, Sfax 3018, Tunisie

B. Jarboui
Institut Supérieur de Commerce et de Comptabilité de Bizerte, Zarzouna 7021, Bizerte, Tunisie

P. Siarry (✉)
LiSSi, Université de Paris 12, 61 avenue du Général de Gaulle, 94010 Créteil, France
e-mail: siarry@univ-paris12.fr

considered as a crucial task in the industrial world. The SDST scheduling problems are frequently encountered in many production systems. They can usually be found in several practical industrial situations, such as in pharmaceutical, metallurgical, chemical industries [3, 9, 26], printing [22], plastics industries [3, 9, 16], car spraying facilities, wafer testing in semiconductor manufacturing, aluminum factories [13], petroleum producing factories and textile companies [9, 17].

The single machine scheduling problem with SDST is a combinatorial problem which was proved to belong to the class of NP-hard problems (see [4]). Several criteria have been presented in the context of the single machine scheduling with SDST. These criteria are essentially: the total weighted tardiness, the weighted sum of squared tardiness, the makespan, the number of tardy jobs, the maximum tardiness, the total flow time, and the total tardiness.

For the total weighted tardiness criterion, some authors are interested in exact methods. Potts and Van Wassenhove [23] presented a Branch-and-Bound (B&B) algorithm. On the other hand, there exist several works which addressed to heuristics and metaheuristics. Liao and Juan [18] presented an Ant Colony Optimization algorithm. For the weighted sum of squared tardiness criterion, Sun et al. [28] developed a Lagrangian relaxation based approach. For the makespan and the number of tardy jobs, Choobineh et al. [2] proposed a multi-objective Tabu Search algorithm. For the maximum tardiness, Luo and Chu [20] developed an algorithm based on B&B permutation schemes. For the total flow time, Bigras et al. [1] presented an exact B&B algorithm.

In this paper, we focused our interest on the total tardiness criterion. Some authors are interested in exact methods. Ragatz [24] developed exact algorithms based on B&B techniques for minimizing total tardiness single machine scheduling with SDST. This algorithm proposed some dominance conditions and a combination of two branching rules best-first and depth-first. Luo and Chu [19] presented another B&B algorithm to solve the same problem. This algorithm comprises the implementation of lower and upper bounding procedures, and dominance elimination rules. Recently, Bigras et al. [1] showed that integer programming formulations of the time-dependent traveling salesman problem can be extended to single machine scheduling problems with SDST. They developed an exact B&B algorithm B&B$_{(\mathrm{BGS})}$, in order to minimize total tardiness and total flow time.

Due to the complexity of scheduling problems, the exact methods are not adequate to solve them. In fact, the NP-hard problems are difficult to solve and no polynomial time algorithm are known for solving them. Moreover, the exact methods are restricted to small instances because of their time requirements. Since these methods are known to be time expensive, so they can not be applied to large instances.

Consequently, several heuristic approaches have been proposed in the literature to solve the single machine scheduling problem with SDST, which aim at obtaining good approximate solutions with respect to solution quality and computational running time.

Some heuristics and metaheuristics have been adapted to treat this problem. Rubin and Ragatz [25] developed a Genetic Search. In this work the authors discussed the utility of Genetic Search to develop near optimal schedules in this environment. Tan and Narasimhan [29] proposed a Simulated Annealing approach and showed that their algorithm outperformed the B&B and Genetic Search designed by Rubin and Ragatz [25]. Tan et al. [30] used four algorithms. Namely, a Random-Start Pairwise Interchange heuristic $\text{RSPI}_{(\text{TNRR})}$, a Genetic Search, a Simulated Annealing and a B&B algorithm. The two first methods were based on the work of Rubin and Ragatz [25]. The Simulated Annealing method was based on that of Tan and Narasimhan [29] and the B&B method was based on that of Ragatz [24]. França et al. [10] developed a Memetic Algorithm based on the hierarchical population. For the local improvement procedure, they developed several neighborhood reduction schemes and proved the effectiveness of these reduction schemes when compared to the complete neighborhood. Gagné et al. [11] developed an Ant Colony Optimization algorithm $\text{ACO}_{(\text{GPG})}$ having a new feature using look-ahead information in the transition rule. They proposed this information in the selection of the next job during the construction of a solution. To increase the performance of their algorithm, the authors introduced a local search algorithm as an improvement phase. Gagné et al. [12] presented a generic approach to find compromise solutions for multiple-objective scheduling problems with SDST. They developed a new hybrid Tabu Search/Variable Neighborhood Search algorithm $\text{TVNS}_{(\text{GGP})}$ for the solution of bi-objective scheduling problems. This approach had three phases: search for the ideal and Nadir points, search for compromise solutions when both total tardiness and setup time were considered and present the pareto-solutions to the decision maker. Choobineh et al. [2] developed a multi-objective Tabu Search. This algorithm had three unique features. It used m-parallel tabu lists, it accounted for the weight of each objective, and it created a bounded solution space. Three criteria were simultaneously optimized, namely, the total tardiness, the makespan and the number of tardy jobs. Gupta and Smith [14] proposed two algorithms: a Problem space-based Local Search heuristic $\text{PLS}_{(\text{GS})}$ and a Greedy Randomized Adaptive Search Procedure $\text{GRASP}_{(\text{GS})}$. Regarding $\text{PLS}_{(\text{GS})}$, they integrated Local Search with respect to both the problem space and the solution space. Concerning $\text{GRASP}_{(\text{GS})}$, they developed a new cost function in the constructive phase, a new hybrid Variable Neighborhood Search for the improvement phase, and Path Relinking based on three different search neighborhoods. Liao and Juan [18] developed another hybrid Ant Colony Optimization algorithm $\text{ACO}_{(\text{LJ})}$ based on a local search procedure for minimizing the total weighted tardiness. The authors introduced the feature of a minimum pheromone value from max-min-ant-system. This algorithm had two distinctive features and was applied to the unweighted version of the problem. Ying et al. [31] developed an Iterated Greedy heuristic $\text{IG}_{(\text{YLH})}$. It is a stochastic Local Search algorithm recently used for combinatorial optimization problems. The objective is to minimize the total weighted and unweighted tardiness of all the jobs.

In this paper, we propose a new approach for solving the SDST problem: a hybrid algorithm which combines two metaheuristics: Greedy Randomized Adaptive Search Procedure (GRASP) proposed by Feo and Resende [5] and Differential Evolution (DE) proposed by Storn and Price [27]. We introduce a Variable Neighborhood Search (VNS) developed by Mladenović and Hansen [21], Hansen and Mladenović [15], for the Local Search phase of GRASP.

The contribution of this paper can be summarized by the introduction of a learning process into the GRASP method. Indeed, this method cannot benefit from the good quality solutions generated during the resolution process. In fact, in order to generate new solutions as new starting points for the Local Search algorithm, GRASP uses fixed probabilities based on a heuristic criterion. Undoubtedly, a heuristic criterion cannot summarize the complexity of a problem and it cannot take into account the particularity of each instance. Moreover, these probabilities are considered as fixed parameters during the resolution phase. In this case, the GRASP method does not benefit from the good solutions generated in the running time. Therefore, it is unable to adjust these probabilities in order to reduce the search space in the most promising regions. In this paper, we attempt to overcome this difficulty by varying these probabilities, in order to ensure the effectiveness of the GRASP method, by introducing a learning process. This process was based on multiplier coefficients that can be evolved into the resolution process. In order to achieve this task, we used the DE algorithm. This choice was not arbitrary: in fact, to guarantee the effectiveness of the learning process with a minimum number of iterations, we needed an algorithm that can assure a fast adjustment of probabilities. Hence, the DE algorithm was used because it is characterized by its fast convergence towards a good quality solution.

When comparing our approach to the results available in the literature, our algorithm obtains good solutions for the single machine scheduling problem with SDST.

The rest of the paper is organized as follows. Section 2 provides a short overview of the single machine scheduling problem with SDST. Section 3 presents a Greedy Randomized Adaptive Search Procedure (GRASP). Section 4 is devoted to the Differential Evolution (DE) approach. Section 5 presents the new GRASP-DE method for solving the single machine scheduling problem with SDST. Section 6 is devoted to the Variable Neighborhood Search (VNS) method. Section 7 deals with computational experiments, results and comments. The paper ends with some conclusions in Sect. 8. In the end of paper, all the notations and the abbreviations are presented in Table 1.

2 Formulation of the single machine scheduling problem with SDST

The single machine scheduling problem with SDST can be described as follows [30]: a set of n jobs are available for processing at time zero on a single machine. All jobs must be completed without interruption. The machine can process only one job at a time. Each job j has a processing time (production duration) PT_j, a due (delivery) date DD_j, and a setup time ST_{ij} which is incurred when job j is undertaken immediately following job i in a job sequence W.

Let W be a sequence of the jobs: $W = [w(0), w(1), \dots, w(n)]$, where $w(j)$ is the index of the jth job in the sequence and $w(0)$ is a dummy job, ($w(0) = 0$) representing the starting setup of the machine. The due date of the jth job in the sequence is denoted by $DD_{w(j)}$, its processing time is denoted by $PT_{w(j)}$, and the setup time between two consecutive jobs ($j - 1$) and j is denoted by $ST_{w(j-1)w(j)}$.

The completion time (or end time) of job j, denoted by $CT_{w(j)}$, is obtained as follows:

$$CT_{w(j)} = \sum_{h=1}^{j} [ST_{w(h-1)w(h)} + PT_{w(h)}] \tag{1}$$

Table 1 The notations and the abbreviations

Notation	Explanation
SDST	Sequence-Dependent Setup Times
B&B	Branch and Bound
B&B$_{(BGS)}$	Branch and Bound algorithm developed by Bigras et al. [1]
RSPI$_{(TNRR)}$	Random-Start Pairwise Interchange heuristic used by Tan et al. [30]
ACO$_{(GPG)}$	Ant Colony Optimization algorithm developed by Gagné et al. [11]
TVNS$_{(GGP)}$	Tabu Search/Variable Neighborhood Search algorithm developed by Gagné et al. [12]
PLS$_{(GS)}$	Problem space-based Local Search heuristic developed by Gupta and Smith [14]
GRASP$_{(GS)}$	Greedy Randomized Adaptive Search Procedure developed by Gupta and Smith [14]
ACO$_{(LJ)}$	Ant Colony Optimization algorithm developed by Liao and Juan [18]
IG$_{(YLH)}$	Iterated Greedy heuristic developed by Ying et al. [31]
GRASP	Greedy Randomized Adaptive Search Procedure
DE	Differential Evolution
VNS	Variable Neighborhood Search
GRASP-DE	Greedy Randomized Adaptive Search Procedure based on Differential Evolution
PT_j	processing time of job j
DD_j	due date of job j
ST_{ij}	setup time between two consecutive jobs i and j
W	sequence of jobs
$w(i)$	the job in the position i in the sequence W
$DD_{w(j)}$	due date of the job $w(j)$
$PT_{w(j)}$	processing time of the job $w(j)$
$ST_{w(j-1)w(j)}$	setup time between two consecutive jobs $w(j-1)$ and $w(j)$
$CT_{w(j)}$	completion time of job $w(j)$ in the sequence
$TR_{w(j)}$	tardiness of job $w(j)$
TT_W	total tardiness of all jobs
RCL_α	Restricted Candidate List
α	threshold parameter of GRASP method
P	population size
F	scaling factor
CR	crossover rate
D	dimension of the vector
x_{ij}	variable j in the ith vector
x_j^U	upper bound of the jth variable
x_j^L	lower bound of the jth variable
r_j	random value uniformly distributed in the range [0,1]
M_i^{k+1}	mutant vector in the generation $k+1$
C_i^{k+1}	trial vector in the generation $k+1$
x_i^k	target vector in the generation k
j_{rand}	random index
f	fitness function
$pr_{w(i)j}$	selection probability of the job j after the job $w(i)$

Table 1 (*Continued*)

Notation	Explanation
$\overline{w}$	remaining jobs after selecting job in position i
g^1	function which attaches more importance to the jobs that have the earliest due date to be scheduled before jobs with the latest due dates
g^2	function which attaches more importance to the setup time
β_1	determine the relative importance of the due date
β_2	determine the relative importance of the setup time
$\eta_{w(i)j}$	probability function of selecting job j after job $w(i)$
π^1	learning factor relative to the due date
π^2	learning factor relative to the setup time
$V_i(x)$	set of solutions in the ith neighborhood of solution x
$N_{\max}$	number of used neighborhood structures
prob	probability of VNS application
$s_{generated}$	solution found by the constructive phase of GRASP
s_{best}	best solution generated by the constructive phase of GRASP
ω	current temperature in the Boltzmann function

The tardiness of the jth job in the sequence is given by:

$$TR_{w(j)} = \max\{0, CT_{w(j)} - DD_{w(j)}\} \tag{2}$$

The objective is to minimize the total tardiness of all the jobs, given by:

$$TT_W = \sum_{j=1}^{n} TR_{w(j)} \tag{3}$$

Consider the following example: (this instance contains 4 jobs)

Let the processing times of the jobs:

20	15	13	18

Let due dates of the jobs:

60	80	20	45

The matrix represents the setup times of the jobs:

$$\begin{array}{c|ccccc}
 & \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\
\hline
\mathbf{1} & 3 & 0 & 4 & 6 & 2 \\
\mathbf{2} & 8 & 3 & 0 & 1 & 3 \\
\mathbf{3} & 5 & 2 & 8 & 0 & 4 \\
\mathbf{4} & 2 & 1 & 7 & 5 & 0
\end{array}$$

The first column represents the setup time of job j if this job is processed in the first position.

Let the sequence $W = \{0, 3, 4, 1, 2\}$, 0 represents the dummy job.

$$w(1) = 3, \quad w(2) = 4, \quad w(3) = 1, \quad w(4) = 2$$

$$CT_{w(0)} = 0$$

$$CT_{w(1)} = CT_{w(0)} + [ST_{w(0)w(1)} + PT_{w(1)}] = 0 + 5 + 13 = 18$$

$$CT_{w(2)} = CT_{w(1)} + [ST_{w(1)w(2)} + PT_{w(2)}] = 18 + 5 + 18 = 41$$

$$CT_{w(3)} = CT_{w(2)} + [ST_{w(2)w(3)} + PT_{w(3)}] = 41 + 2 + 20 = 63$$

$$CT_{w(4)} = CT_{w(3)} + [ST_{w(3)w(4)} + PT_{w(4)}] = 63 + 3 + 15 = 81$$

$$TR_{w(1)} = \max\{0, CT_{w(1)} - DD_{w(1)}\} = \max\{0, 18 - 20\} = \max\{0, -2\} = 0$$

$$TR_{w(2)} = \max\{0, CT_{w(2)} - DD_{w(2)}\} = \max\{0, 41 - 45\} = \max\{0, -4\} = 0$$

$$TR_{w(3)} = \max\{0, CT_{w(3)} - DD_{w(3)}\} = \max\{0, 63 - 60\} = \max\{0, 3\} = 3$$

$$TR_{w(4)} = \max\{0, CT_{w(4)} - DD_{w(4)}\} = \max\{0, 81 - 80\} = \max\{0, 1\} = 1$$

Hence, the total tardiness is equal to 4.

$$TT_W = TR_{w(1)} + TR_{w(2)} + TR_{w(3)} + TR_{w(4)} = 0 + 0 + 3 + 1 = 4$$

3 Greedy randomized adaptive search procedure

The Greedy Randomized Adaptive Search Procedure (GRASP) methodology was first introduced by Feo and Resende [5]. Feo et al. [6] used GRASP for a difficult single machine scheduling problem. Feo et al. [7] applied GRASP to develop an efficient heuristic for the single machine scheduling problem with sequence dependent setup costs and linear delay penalties. GRASP is a procedure of "multi-start" research for finding approximate solutions to combinatorial optimization problems. This method consists of setting up a realizable initial solution, further improved by a Local Search. These two stages are iteratively repeated and the best found solution is finally kept.

In the first stage, the solution is created step by step by adding elements to a partial solution. The addition of each element is guided by a greedy evaluation. Each element is arbitrarily selected from a Restricted Candidate List (RCL$_\alpha$) containing the most promising elements according to the greedy evaluation, where α is a threshold parameter controlling the balance between greedy and randomized construction. The solutions produced with the construction phase are not guaranteed to be locally optimal. Thus, a Local Search method is applied as a second stage in order to improve the solution produced in the construction phase.

The Local Search method iteratively replaces the current solution with a better solution, if any, belonging to its neighborhood. The Local Search halts by producing a local optimum. Generally, GRASP is stopped either when a given number of iterations is reached or when no further improvements are possible.

We can observe that the GRASP approach does not benefit from the good solutions obtained during the resolution phase since it does not carry out any adjustment of probability during this phase. On the contrary, there are other approaches that take advantage of such a mechanism like the ACO algorithm that adjusts the probabilities according to the results obtained. For this reason, we will propose, in this framework, a similar mechanism that draws on the DE algorithm.

4 Differential evolution

Differential Evolution (DE) is a type of evolutionary algorithm introduced by Storn and Price [27] for optimization problems. It has been shown to be an effective, fast and robust optimization method. DE algorithm belongs to the family of evolutionary computation and makes use of the concept of fitness. Three important factors must be necessarily considered when the DE algorithm is used: the population size is adjusted by the real parameter P, the scaling factor F, commonly called the "amplification factor", and the crossover rate *CR*. The first factor P should not be too small so as to avoid stagnation and to provide sufficient exploration. The second factor F is one of the control parameters that significantly affects the convergence rate. The third factor *CR* is a user-defined value that controls the fraction of parameter values that are copied from the mutant vector. Each element in the population is real-valued vector with dimension D. DE algorithm mainly consists of four steps or phases: initialization, mutation, recombination and selection. Note that the used version illustrated in this framework is "DE/rand/1/bin" because the vector is randomly chosen.

In the initialization phase, each variable j in the ith vector, denoted by x_{ij}, is initialized within its boundaries x_j^U and x_j^L, which indicate the upper and lower bound of the jth variable respectively, by means of the following equation:

$$x_{ij} = x_j^L + r_j(x_j^U - x_j^L) \tag{4}$$

where r_j is a random value uniformly distributed in the range [0, 1], $j = 1, 2, \dots, D$ and $i = 1, 2, \dots, P$.

In the mutation phase, DE algorithm generates a mutant vector (M_i^{k+1}), by adding a weighted difference of two vectors, x_{t1j}^k and x_{t2j}^k, selected randomly from the current population k to a third vector x_{t0j}^k ($t_0 \neq t_1 \neq t_2$), as shown in the following equation:

$$M_{ij}^{k+1} = x_{t0j}^k + r_j \times F(x_{t1j}^k - x_{t2j}^k) \tag{5}$$

The scaling factor F is a real number that controls the amount of perturbation added to the parent vector x_{t0j}^k. F is usually kept between 0 and 2.

In the crossover phase, the crossover operation generates a new vector, called trial vector (C_i^{k+1}), by crossing the mutated vector M_i^{k+1} and the target vector x_i^k:

$$C_{ij}^{k+1} = \begin{cases} M_{ij}^{k+1} & \text{if } r_j \leq CR \text{ or } j = j_{rand} \\ x_{ij}^k & \text{otherwise} \end{cases} \tag{6}$$

where j_{rand} is a random index selected in $[1, D]$ to assure that the trial vector does not duplicate the target vector. The crossover rate, $CR \in [0, 1]$, is a control parameter which allows escaping from local optima.

In the selection phase, we compare the costs of the trial vector and the target vector. Hence, this phase decides which vector is retained. It determines survivors by pairwise comparison.

$$x_i^{k+1} = \begin{cases} C_i^{k+1} & \text{if } f(C_i^{k+1}) \leq f(x_i^k) \\ x_i^k & \text{otherwise} \end{cases} \tag{7}$$

where f is the fitness function.

The entire procedure is repeated over numerous generations until the optimum is found, or a selected stopping condition is satisfied.

5 Constructive phase of GRASP based on DE algorithm for solving the single machine scheduling problem with SDST

In this section, we will introduce an approach in order to ameliorate the performance of the GRASP method through the introduction of a learning process in its construction phase. At first, we will present the constructive phase of the GRASP method adopted for the SDST problem. Then, we will present our approach proposed for the amelioration of this phase based on the DE algorithm.

In this paper, the RCL is composed of jobs selected according to a threshold parameter $\alpha \in [0, 1]$ applied to their selection probability $pr_{w(i)j}$. In other words, the number of jobs in the list is limited by using the parameter α and all jobs with a probability greater than or equal to $pr_{\max} * \alpha$ are introduced in the list.

Let DD_i denotes the due date of the job i, PT_i denotes the processing time of the job i, $\overline{w}$ denotes the remaining job after selecting job in position i, and ST_{ij} denotes the setup time which is incurred when job j is undertaken immediately following job i in a sequence.

The selection probability $pr_{w(i)j}$ of the job j after the job $w(i)$ is given by the following formula:

$$pr_{w(i)j} = \frac{g_j^1 g_{w(i)j}^2}{\sum_{k \in \overline{w}} g_k^1 g_{w(i)k}^2} \quad \forall j \in \overline{w} \tag{8}$$

where g^1 and g^2 are given by the following formulas:

$$g_j^1 = \left(\frac{1}{(DD_j - \min_{k=1,\dots,n} DD_k) + 1} \right)^{\beta_1} \tag{9}$$

$$g_{w(i)j}^2 = \left(\frac{1}{PT_{w(i)} + ST_{w(i)j}} \right)^{\beta_2} \tag{10}$$

β_1 and β_2 determine the relative importance of the due date and the setup time, respectively. Note that the decrease of the due date value corresponds to an increase of the function g^1. Let $pr_{w(0)i}$ be the selection probability of the job i when there is no job to be scheduled in the sequence. Hence, $PT_{w(0)} = 1$ denotes the processing time of a dummy job.

The function g^1 attaches more importance to the jobs that have the earliest due dates to be scheduled before jobs with the latest due dates. The function g^2 depends on the value of $PT_{w(i)} + ST_{w(i)j}$ that can be considered as the distance between job j and $w(i)$ when we consider the problem as a traveling salesman problem. The function g^2 attaches more importance to the setup time. While the job $w(i)$ is scheduled, so the job that has the highest probability to be scheduled is the job j that has the smallest setup time before job $w(i)$.

In what follows, we will propose the suggested approach (GRASP-DE) to improve the performance of GRASP method. This approach is based on the introduction of a learning process which will help us to adjust the probabilities $pr_{w(i)j}$ in order to ameliorate the starting points created in the constructive phase, which will be used by the Local Search algorithm.

For this purpose, we multiply g_j^1 by the learning factor π_j^1 and we multiply $g_{w(i)j}^2$ by the learning factor $\pi_{w(i)j}^2$. These two learning factors have similar features of pheromone information in the ACO algorithm.

The parameters π^1 and π^2 aim to adjust the functions g^1 and g^2 according to the importance of the due date and the setup time respectively for each job and aim to focus the search in regions containing high quality solutions. These two factors are generated and adjusted by the DE algorithm. During the runtime, DE algorithm adjusts π^1 and π^2 values using previously generated solutions by selecting parameters that have contributed to good solutions. The evolution process of the DE algorithm is based on the evaluation of schedules generated by a new probability function $\eta_{w(i)j}$.

Presentation and creation of the new solution:

$$\eta_{w(i)j} = \frac{g_j^1 \pi_j^1 g_{w(i)j}^2 \pi_{w(i)j}^2}{\sum_{k \in \overline{w}} g_k^1 \pi_k^1 g_{w(i)k}^2 \pi_{w(i)k}^2} \quad \forall j \in \overline{w} \tag{11}$$

where $\eta_{w(i)j}$ is the probability of selecting job j after job $w(i)$, $\pi_{w(i)j}^2$ is the learning factor relative to the setup time, and π_j^1 is the learning factor relative to the due date.

$$\pi^1 = (\pi_1^1, \pi_2^1, \ldots, \pi_n^1)$$

$$\pi^2 = \begin{pmatrix} 0 & \pi_{12}^2 & \cdots & \pi_{1n}^2 \\ \pi_{21}^2 & 0 & \cdots & \pi_{2n}^2 \\ \vdots & \vdots & \ddots & \vdots \\ \pi_{n1}^2 & \pi_{n2}^2 & \cdots & 0 \end{pmatrix}$$

A candidate solution is composed of π^1 and π^2 containing continuous values within the range [0, 1]. To generate an initial solution, we randomly choose continuous values uniformly distributed in the range [0, 1].

These two learning factors will be optimized through the DE algorithm. When a new individual is created, since the values of the corresponding vector π^1 and matrix

π^2 are continuous, the DE algorithm becomes unsuitable to evaluate the solution, which brings us to use the constructive phase of GRASP.

The general procedure of the learning process can be presented as follows: first, at each iteration of the GRASP algorithm, a new vector π^1 and a new matrix π^2 will be created by the algorithm based on (5) and (6). Then a sequence is created by the constructive phase of GRASP method based on the vector π^1, the matrix π^2 and the probability function $\eta_{w(i)j}$. The fitness of this sequence will be used to evaluate the new solution generated by the DE algorithm in the selection phase. Each solution generated might be the object of an application of the VNS algorithm. The decision to apply VNS will be based on the evaluation of the solution generated. The details of the decision criterion will be presented in the next section.

6 Variable neighborhood search

In this section, we will present the VNS algorithm used in the amelioration phase of the GRASP approach. In fact, the use of a simple Local Search would not let us carry out a good exploration of the neighborhood of the solution generated during the construction phase. Hence, the use of a more sophisticated approach can help us avoid falling into the first local optimum encountered. This will increase the efficiency of our algorithm.

VNS was originally introduced by Mladenović and Hansen [21], Hansen and Mladenović [15]. It is used for solving combinatorial and global optimization problems. VNS allows systematic changes of several neighborhood structures within a Local Search procedure. At first, a finite set of pre-selected neighborhood structures is predefined ($V_i,\ i = 1, 2, \dots, N_{\max}$), where $V_i(x)$ is the set of solutions in the ith neighborhood of solution x and $N_{\max}$ is the number of used neighborhood structures. An initial solution is found, and a stopping criterion is chosen. Initially i is set to 1 and in the *shaking* step a solution x' is randomly generated in $V_i(x)$. Then, a *Local Search* method is applied to x' to attain a local optimum x''. If x'' is better than x, then x is replaced with x'' and i is set again to 1. Otherwise, the neighborhood size i is incremented in one unit to enlarge the scope of the search. The method repeats iteratively *shaking* and *Local Search* phases until a stopping condition is reached. This stopping criterion may be the maximum number of iterations, the maximum allowed CPU time, or the maximum number of iterations between two improvements.

In our algorithm, we use VNS as the improvement procedure in GRASP [8]. The generated solutions in the constructive phase of GRASP are not all well evaluated. Then, we apply the VNS algorithm just to the best solutions. Let *prob* denote the probability of VNS application. This probability is inspired from the Boltzmann function:

$$prob = \exp\left(\frac{(f(s_{generated}) - f(s_{best}))/f(s_{best})}{\omega}\right) \tag{12}$$

where $s_{generated}$ denotes the solution found by the constructive phase of GRASP, s_{best} denotes the best solution generated by the algorithm, and ω represents the current temperature in the Boltzmann function.

In the proposed local search method we use two neighborhood structures, namely the *permutation* and the *insertion*. The permutation procedure consists in swapping all jobs pairwise. The insertion procedure starts when the local optimum is reached and all the swaps are executed. If a local optimum is found again, we reapply the permutation procedure. The improved solution is considered as the generated solution for the first procedure. This strategy is repeated until no improvement.

Before executing the Local Search algorithm, if the reached solution is better than the best solution, then it is replaced by the new solution and i is set to 1. Otherwise, the neighborhood size i is incremented in one unit. Thereafter, we go to the shaking phase by applying i swapping moves, randomly chosen from the best solution, and we repeat the steps described above, until the maximal number of iterations is reached.

The general framework of this algorithm can be presented as follows:

```
Begin VNS algorithm (Set $i = 1$)

Step 1: apply $i$ swapping moves in $x$ to obtain $x'$;
Step 2: apply local search procedure to $x'$ to obtain $x''$;
Step 3: if $x''$ is better than $x$ then $x = x''$ and $i = 1$; else $i = i + 1$;
Step 4: if $i \leq N_{\max}$ then go to step 1; else go to step 5;
Step 5: if the stopping criterion is not satisfied then $i = 1$ and go to step 1; else END
  of algorithm;
```

The pseudo code of the GRASP-DE algorithm is given in Fig. 1.

7 Computational results

This section reports computational experiments that evaluate the effectiveness of our GRASP-DE algorithm and compare its performance against those of various algorithms proposed in the literature.

The algorithm was coded in C++. The instance problems were run in Windows XP on desktop PC with Intel Pentium IV, running at 3.2 GHz processor with 512 MB of RAM memory.

First, our algorithm was tested, for "small size" problems, on 32 benchmark problem instances. The test problem set was divided by size into four sets having 15, 25, 35, 45 jobs, respectively. These problems are initially proposed by Rubin and Ragatz [25] and are available at http://www.msu.edu/~rubin/research.html.

Then, the same algorithm was performed for "large size" problems on 32 benchmark problem instances. The test problem set was divided by size into four sets having 55, 65, 75, 85 jobs, respectively. These problems are provided by Gagné et al. [11] and are available at http://wwwdim.uqac.ca/~c3gagne/home_fichiers/ProbOrdo.htm.

7.1 Parameters setting

One of the problems that can be encountered in the experimentation is the definition of parameters. An exhaustive study of all possible parameters combinations is a

```
Initialize the population of $P$ individuals by using (4);
loop
    for $i = 1, 2, \ldots, P$ do
        //Step 1: create new learning factors by DE
        select the 3 individuals at random from the population;
        create a new offspring composed of the learning factors
        $\pi^1$ and $\pi^2$ using (5) and (6);
        //Step 2: create a new sequence using the constructive
        phase of GRASP
        create a new sequence $W$ using the constructive phase of
        GRASP based on (11);
        //Step 3: update the population of DE using the evaluation
        of the sequence generated by the
        //constructive phase of GRASP;
        if the fitness $f(W)$ of the new sequence $W$ is better than
        the fitness $f_i$ of the individual $i$ then
            replace the individual $i$ with the new offspring;
            $f_i$ takes the value of $f(W)$;
        endif
        //Step 4: apply VNS algorithm to the good solution
        compute the probability prob using $f(W)$ and (12);
        generate a random value $r$ uniformly in the range [0, 1];
        if $r$ is lower than prob then
            apply VNS to the new sequence $W$;
        endif
        if the new sequence $W$ is better than the best sequence $W^*$ then
            $W^*$ takes $W$;
        endif
    endfor
until (stopping criterion is reached)
```

Fig. 1 GRASP-DE procedure

crucial task. For example, to test the combination of the following parameters:

$$\beta_1 = \{0.5, 0.6, 0.7, 0.8, 0.9, 1\}, \qquad \beta_2 = \{0.5, 0.6, 0.7, 0.8, 0.9, 1\},$$
$$P = \{10, 20, 30, 40, 50, 60\}, \qquad F = \{1, 1.25, 1.50, 1.75, 2\},$$
$$CR = \{0.7, 0.75, 0.8, 0.85, 0.9, 0.95\} \quad \text{and} \quad \alpha = \{0.6, 0.7, 0.8, 0.9, 0.95\},$$

we must test our algorithm 32400 times for all instances. In order to solve this problem, we will suppose that the obtained result by each parameter is independent of other parameters, and we will test all possible values of each parameter by fixing other parameters. Initially, all parameters are set arbitrarily. Then, each parameter will be defined according to the best values obtained by other parameters. This procedure can be applied in an iterative way, as a local search procedure. It will stop until convergence to an unchanged combination of parameters. Since the execution time required to test all instances is very important, we will limit to a single iteration. The parameters of our algorithm were set after a series of extensive experiments based on the best found results, the given parameters are as follows: $\beta_1 = 0.5$, $\beta_2 = 1$, $P = 40$, $F = 2$, $CR = 0.8$ and $\alpha = 0.95$.

Regarding VNS algorithm, the best parameters obtained by our study are

$$\omega = \frac{(f(s_{generated}) - f(s_{best}))/f(s_{best})}{\log(prob)} = \frac{0.01}{\log(0.5)}$$

(the given value ω indicates that we can accept the application of VNS to the generated solution with a deviation from the best solution equal to 1%, with probability 0.5) and the maximum number of iterations of VNS was set to 20 without improvement.

Each problem instance was run 20 times. To perform a comparative study with other approaches, we set a maximal computational time equal to $n^2/5$ seconds as a stopping criterion. This parameter guaranteed that our computational time is lower than the computational time of last GRASP algorithm developed by Gupta and Smith [14] multiplied by the ratio between their and our processor speeds.

7.2 Comparative study

We compared the best solution obtained from our GRASP-DE algorithm and other algorithms found in the literature on the small and large problem sets. Table 2 shows the comparison among RSPI$_{\text{(TNRR)}}$ algorithm used by Tan et al. [30], ACO$_{\text{(GPG)}}$ algorithm proposed by Gagné et al. [11], TVNS$_{\text{(GGP)}}$ algorithm proposed by Gagné et al. [12], PLS$_{\text{(GS)}}$ algorithm proposed by Gupta and Smith [14], ACO$_{\text{(LJ)}}$ algorithm proposed by Liao and Juan [18], GRASP$_{\text{(GS)}}$ algorithm proposed by Gupta and Smith [14], IG$_{\text{(YLH)}}$ algorithm proposed by Ying et al. [31], B&B$_{\text{(BGS)}}$ algorithm proposed by Bigras et al. [1] and GRASP-DE, on the small test problems. Table 3 shows the comparison among ACO$_{\text{(GPG)}}$ algorithm proposed by Gagné et al. [11], TVNS$_{\text{(GGP)}}$ algorithm proposed by Gagné et al. [12], PLS$_{\text{(GS)}}$ algorithm proposed by Gupta and Smith [14], ACO$_{\text{(LJ)}}$ algorithm proposed by Liao and Juan [18], GRASP$_{\text{(GS)}}$ algorithm proposed by Gupta and Smith [14], IG$_{\text{(YLH)}}$ algorithm proposed by Ying et al. [31] and GRASP-DE, on the large test problems. However, the results of RSPI$_{\text{(TNRR)}}$ algorithm used by Tan et al. [30] are not available for the large size problems.

Compared to the previous results available in the literature, our algorithm found 12 better solutions (these solutions are presented in shaded area) for the single machine scheduling problem with SDST.

The B&B column (Table 2) is the Branch-and-Bound solution as stated in Tan et al. [30]. As shown in Tables 2 and 3, the TT columns indicate the total tardiness criterion.

For the first set, with 15 jobs of small size problems, all the algorithms found the optimal solution. The results in Table 2 show that all algorithms provide optimal solutions for the 11 instances whose optimal solutions have zero tardiness. Table 3 shows that all algorithms provide optimal solutions for the 10 instances whose optimal solutions have zero tardiness, except for Prob555 instance problem, where PLS$_{\text{(GS)}}$ algorithm provides a better solution equal to 4.

The B&B$_{\text{(BGS)}}$ algorithm proposed by Bigras et al. [1] was able to solve optimally all the small test problems with the exception of five instances: Prob604, Prob703, Prob704, Prob707, Prob708. In order to solve these open instances, Bigras et al. [1] run their algorithm with the bound deduced by the most efficient developed approach.

Table 2 Quality solution comparison of different algorithms for small size problems

		Best solution	B&B	RSPI(TNRR)		ACO(GPG)		TVNS(GGP)	PLS(GS)		ACO(LJ)	IG(YLH)	B&B(BGS)	GRASP(GS)		GRASP-DE							
		TT	TT	TT[c]		TT[c]		TT[c]	TT[c]		TT[c]	TT[c]	TT[c]	TT[c]		TT[c]				TIME			
Problems	Jobs	sol	sol	min	max	min	max	min	min	max	min	min	min	min	max	min	avg	max	st.dev	min	avg	max	st.dev
Prob401	15	90[a]	90[a]	0.0[a]	3.3	0.0[a]	7.8	0.0[a]	0.0[a]	4.44	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.05	0.17	0.39	0.10
Prob402	15	0[a]	0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.02	0.02	0.03	0.01
Prob403	15	3418[a]	3418[a]	0.0[a]	0.2	0.0[a]	2.1	0.0[a]	0.0[a]	0.23	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.03	0.03	0.05	0.01
Prob404	15	1067[a]	1067[a]	0.0[a]	12.7	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.02	0.04	0.05	0.01
Prob405	15	0[a]	0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.00	0.00	0.00	0.00
Prob406	15	0[a]	0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.00	0.02	0.03	0.01
Prob407	15	1861[a]	1861[a]	0.0[a]	0.4	0.0[a]	1.1	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.03	0.04	0.05	0.01
Prob408	15	5660[a]	5660[a]	0.0[a]	0.9	0.0[a]	1.5	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.03	0.04	0.08	0.01
Prob501	25	261[a]	264	0.8	1.5	−1.1[a]	1.9	−1.1[a]	1.14[a]	1.52	−0.38	−1.13[a]	−1.14[a]	−1.1[a]	−0.4	−1.14[a]	−1.14	−1.14	0.0	0.19	1.56	5.78	1.49
Prob502	25	0[a]	0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.00	0.06	0.06	0.02
Prob503	25	3497[a]	3511	−0.4[a]	−0.4	−0.4[a]	0.9	−0.2	−0.4[a]	0.06	−0.4[a]	−0.4[a]	−0.4[a]	−0.4[a]	−0.4	−0.4[a]	−0.4	−0.4	0.0	0.13	0.96	3.03	0.85
Prob504	25	0[a]	0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.06	0.08	0.09	0.01
Prob505	25	0[a]	0[a]	0.0[a]	[b]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.06	0.08	0.13	0.01
Prob506	25	0[a]	0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.06	0.07	0.08	0.01
Prob507	25	7225[a]	7225[a]	0.0[a]	0.2	0.7	3.7	0.0[a]	0.0[a]	0.54	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.14	0.65	1.94	0.50
Prob508	25	1915[a]	2067	−7.4[a]	−7.4	−5.9	14.2	−7.4[a]	−7.35[a]	−7.35	−7.35[a]	−7.35[a]	−7.35[a]	−7.4[a]	−7.4	−7.35[a]	−7.35	−7.35	0.0	0.11	0.14	0.20	0.02

Table 2 (*Continued*)

		Best solution	B&B	RSPI$_{(TNRR)}$		ACO$_{(GPG)}$		TVNS$_{(GGP)}$	PLS$_{(GS)}$		ACO$_{(LJ)}$	IG$_{(YLH)}$	B&B$_{(BGS)}$	GRASP$_{(GS)}$		GRASP-DE							
		TT	TT	TT[c]		TT[c]		TT[c]	TT[c]		TT[c]	TT[c]	TT[c]	TT[c]		TT[c]				TIME			
Problems	Jobs	sol	sol	min	max	min	max	min	min	max	min	min	min	min	max	min	avg	max	st.dev	min	avg	max	st.dev
Prob601	35	12[a]	30	13.3	46.7	−46.7	6.7	−60.0[a]	0.0	33.33	−53.33	−60.0[a]	−60.0[a]	−46.7	−3.3	−56.67	−44.67	−36.67	5.13	2.56	65.59	223.58	78.42
Prob602	35	0[a]	0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.14	0.16	0.25	0.03
Prob603	35	17587[a]	17774	−0.6	0.7	−0.5	0.3	−1.0[a]	−0.76	−0.14	−0.68	−1.05[a]	−1.05[a]	−1.1[a]	−0.8	−1.05[a]	−1.04	−0.95	0.04	0.92	64.06	220.82	59.53
Prob604	35	19092[a]	19277	−0.6	0.1	−0.8	1.3	−0.6	−0.81	0.08	−0.96[a]	−0.96[a]	−0.96[a]	−0.9	−0.6	−0.96[a]	−0.96	−0.96	0.0	1.34	22.44	79.93	19.45
Prob605	35	228[a]	291	−8.2	−2.1	−15.1	−1.0	−21.6[a]	−5.5	−0.69	−17.53	−21.65[a]	−21.65[a]	−16.5	−13.4	−21.65[a]	−19.48	−18.21	1.0	1.30	48.89	228.71	69.71
Prob606	35	0[a]	0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.13	0.15	0.19	0.02
Prob607	35	12969[a]	13274	−1.4	−0.2	−1.4	0.6	−2.3[a]	−2.2	−1.23	−1.99	−2.3[a]	−2.3[a]	−2.3[a]	−1.8	−2.3[a]	−2.3	−2.3	0.0	2.39	13.50	34.75	7.46
Prob608	35	4732[a]	6704	−29.3	−29.0	−29.4[a]	−20.1	−29.4[a]	−29.42[a]	−29.01	−29.42[a]	−29.42[a]	−29.42[a]	−29.4[a]	−29.4	−29.42[a]	−29.42	−29.42	0.0	0.42	0.66	1.53	0.26
Prob701	45	97[a]	116	12.9	28.4	−11.2	0.0	−15.5	0.0	18.1	−11.21	−11.21	−16.38[a]	−11.2	2.6	−11.21	−8.84	−6.9	1.4	4.52	39.11	128.64	36.96
Prob702	45	0[a]	0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.25	0.30	0.39	0.03
Prob703	45	26506[a]	27097	−0.6	0.1	−1.6	−0.5	−2.2[a]	−1.38	−0.62	−1.95	−2.22	−2.08	−1.8	−1.3	−2.18[a]	−2.17	−2.04	0.03	25.50	178.63	381.57	116.85
Prob704	45	15206[a]	15941	−3.5	−1.8	−2.8	−0.3	−4.6[a]	−4.57	−3.84	−3.34	−4.61[a]	3.99	−4.6[a]	−4.4	−4.61[a]	−4.61	−4.61	0.0	1.55	6.16	22.66	5.76
Prob705	45	200[a]	234	14.1	23.1	−5.1	15.4	−14.5[a]	5.13	16.24	−6.41	−14.53[a]	−14.53[a]	−5.1	6.8	−11.11	−7.84	−5.13	1.47	2.80	69.50	298.33	77.71
Prob706	45	0[a]	0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0[a]	0.0[a]	0.0	0.0[a]	0.0[a]	0.0[a]	0.0[a]	0.0	0.0[a]	0.0	0.0	0.0	0.27	0.29	0.34	0.02
Prob707	45	23789[a]	25070	−3.5	−2.6	−4.2	−2.9	−5.0	−4.49	−3.53	−4.54	−5.09	−5.08	−5.0	−4.4	**−5.11**[a]	−5.04	−4.91	0.06	8.84	176.10	394.95	130.27
Prob708	45	22807[a]	24123	−4.7	−3.3	−3.2	−0.6	−5.2	−5.46[a]	−4.81	−4.54	−5.46[a]	−5.36	−5.5[a]	−5.3	−5.46[a]	−5.46	−5.46	0.0	1.45	7.33	26.46	6.28
Total Time																				55.31	696.83	2055.09	611.83

[a]Indicates the optimal solution

[b]Divided by zero, the worst solution is 6.0

[c]The results are given as percentage from the Branch-and-Bound solution as reported in Tan et al. [30]

Table 3 Quality solution comparison of different algorithms for large size problems

		Best solution	ACO$_{(GPG)}$				TVNS$_{(GGP)}$	PLS$_{(GS)}$			ACO$_{(LJ)}$	IG$_{(YLH)}$	GRASP$_{(GS)}$			GRASP-DE							
		TT	TT				TT	TT			TT	TT	TT			TT				TIME			
Problems	Jobs	sol	min	avg	max	st.dev	min	min	avg	max	min	min	min	avg	max	min	avg	max	st.dev	min	avg	max	st.dev
Prob551	55	183	212	241.3	273	18.6	185	268	300.15	313	183[a]	183[a]	242	260.4	269	215	220.25	230	4.76	4.30	59.94	294.11	72.22
Prob552	55	0	0	0.0	0	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.47	0.49	0.61	0.03
Prob553	55	40540	40828	41109.9	41303	116.8	40644	41077	41173.7	41287	40676	40598	40678	40835.6	40927	**40540**[a]	40603.15	40658	31.14	16.01	332.74	589.67	193.45
Prob554	55	14653	15091	15621.3	16423	320.9	14711	14724	14828.4	14961	14684	14653[a]	14653[a]	14655.7	14675	14653[a]	14653	14653	0.00	5.50	24.08	57.92	13.79
Prob555	55	0	0	2.1	12	3.8	0	4	12.9	20	0	0	0	0.7	3	0	0	0	0.00	1.45	2.40	5.91	1.10
Prob556	55	0	0	0.0	0	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.45	0.48	0.56	0.03
Prob557	55	35813	36489	37308.6	37973	374.4	35841	36128	36284.6	36464	36420	35827	35883	35972.1	36066	**35813**[a]	35835.55	35848	7.92	75.44	310.07	550.23	172.11
Prob558	55	19871	20624	21386.7	22457	427.1	19872	19871[a]	20015.6	20185	19888	19871[a]	19871[a]	19871.4	19880	20078	20078	20078	0.00	2.70	30.90	149.32	34.03
Prob651	65	268	295	319.1	350	16.3	268[a]	343	367.85	379	268[a]	268[a]	333	355.7	368	288	300.10	309	5.88	9.27	145.24	789.38	168.83
Prob652	65	0	0	0.0	0	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.73	0.82	1.00	0.07
Prob653	65	57569	57779	58266.8	58653	223.3	57602	58298	58506.3	58703	57584	57584	57880	58097.1	58276	**57569**[a]	57669	57780	55.55	48.79	411.34	805.65	227.08
Prob654	65	34301	34468	35365.4	36107	419.0	34466	34739	34860.9	35008	34306	34306	34410	34522.5	34628	**34301**[a]	34314.45	34398	29.78	4.82	413.18	830.19	272.49
Prob655	65	2	13	25.3	38	6.2	2[a]	39	46.35	51	7	2[a]	30	34.95	41	11	17.35	22	2.64	10.54	162.15	809.65	191.54
Prob656	65	0	0	0.0	0	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0.75	0.80	0.91	0.04
Prob657	65	54895	56246	57027.5	57825	470.0	55080	55630	55831.7	56053	55389	55080	55355	55473.6	55612	**54895**[a]	55065.05	55180	72.34	50.16	446.24	845.03	265.52
Prob658	65	27114	29308	30155.9	31074	524.2	27187	27186	27417.2	27642	27208	27114[a]	27114[a]	27130.6	27164	27114[a]	27114	27114	0.00	7.55	60.75	148.02	42.83

Table 3 (*Continued*)

		Best solution	$ACO_{(GPG)}$				$TVNS_{(GGP)}$	$PLS_{(GS)}$			$ACO_{(LJ)}$	$IG_{(YLH)}$	$GRASP_{(GS)}$			GRASP-DE							
		TT	TT				TT	TT			TT	TT	TT			TT				TIME			
Problems	Jobs	sol	min	avg	max	st.dev	min	min	avg	max	min	min	min	avg	max	min	avg	max	st.dev	min	avg	max	st.dev
Prob751	75	241	263	311.6	368	23.2	241[a]	350	370.45	388	241[a]	[b]	317	334.05	347	253	272.15	284	6.56	11.23	272.07	793.41	223.28
Prob752	75	0	0	0.0	0	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	1.13	1.18	1.28	0.05
Prob753	75	77663	78211	78604.1	79088	233.7	77739	78599	78946.6	79270	77663[a]	77663[a]	78211	78689.4	78859	77787	77891.20	78013	65.93	75.64	591.11	1087.84	327.16
Prob754	75	35200	35826	37514.3	38333	492.5	35709	35730	36085.3	36256	35630	35250	35323	35413.6	35487	**35200**[a]	35243.40	35312	35.83	25.30	587.55	1125.04	351.62
Prob755	75	0	0	0.0	0	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	1.17	1.41	2.28	0.25
Prob756	75	0	0	0.0	0	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	1.13	1.21	1.36	0.08
Prob757	75	59735	61513	62216.8	63284	442.4	59763	60668	60853	61075	60108	59763	60217	60452.6	60556	**59735**[a]	59910.45	60008	70.43	28.81	503.09	1006.60	347.78
Prob758	75	38339	40277	42018.5	42964	806.4	38789	38515	38858.8	39118	38704	38431	38368	38456.8	38548	**38339**[a]	38347.05	38362	11.26	18.84	393.74	1080.24	320.29
Prob851	85	384	453	511.0	557	30.2	384[a]	563	601.25	619	455	390	531	559.15	579	435	450.65	467	8.22	21.31	417.51	1059.32	308.21
Prob852	85	0	0	0.0	0	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	1.56	1.62	1.86	0.08
Prob853	85	97642	98540	98949.0	99250	212.8	97880	99262	99639.5	99885	98443	97880	98794	99118.6	99296	**97642**[a]	98009.10	98189	123.50	70.97	674.06	1257.95	349.98
Prob854	85	79278	80693	81702.6	82728	567.0	80122	80528	81199.6	81585	79553	79631	80338	80695.5	80962	**79278**[a]	79565.55	79750	121.01	43.70	731.49	1393.13	471.89
Prob855	85	283	333	373.5	409	21.7	283[a]	425	452.7	469	324	283[a]	393	417.45	436	315	333.65	344	7.19	18.88	381.03	710.85	242.20
Prob856	85	0	0	0.0	0	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0.00	1.53	1.60	1.73	0.05
Prob857	85	87244	89654	90569.2	91447	501.8	87244[a]	88547	89002.3	89323	87504	87244[a]	88089	88402.1	88598	87320	87509.90	87695	108.01	91.64	804.64	1430.15	477.30
Prob858	85	74785	77919	79299.5	80612	689.4	75533	75620	75837	76046	75506	75029	75217	75401.9	75517	**74785**[a]	74885.35	74962	49.59	37.84	641.65	1323.69	446.03
Total Time																				689.61	8406.58	18154.89	5521.41

[a]Is the best result among all algorithms

[b]The result is not mentioned by the author

The CPU time was not considered by Bigras et al. [1]. Our algorithm GRASP-DE solved to optimality all the small test problems with the exception of three instances: Prob601, Prob701, and Prob705. So, our GRASP-DE solved 90.63% of all the small instances. The $\mathrm{IG}_{(\mathrm{YLH})}$ algorithm solved 90.63% of all the small instances. The $\mathrm{B\&B}_{(\mathrm{BGS})}$ algorithm solved 87.5% of all the small instances. The $\mathrm{TVNS}_{(\mathrm{GGP})}$ algorithm solved 84.38% of all the small instances. The $\mathrm{GRASP}_{(\mathrm{GS})}$ algorithm solved 78.13% of all the small instances. The $\mathrm{PLS}_{(\mathrm{GS})}$ algorithm solved 68.75% of all the small instances. The $\mathrm{ACO}_{(\mathrm{LJ})}$ algorithm solved 65.63% of all the small instances. The $\mathrm{ACO}_{(\mathrm{GPG})}$ algorithm solved 59.38% of all the small instances. The $\mathrm{RSPI}_{(\mathrm{TNRR})}$ algorithm solved 59.38% of all the small instances. We can remark that our algorithm and the $\mathrm{IG}_{(\mathrm{YLH})}$ algorithm are the most competitive algorithms among the others, for the small instance problems. Our algorithm found the best result for the instance "Prob707" compared with all other algorithms. We note that this instance is solved optimally by our algorithm GRASP-DE for the first time.

For large instances, we can see that our algorithm is able to improve the best known solutions for 11 of the 32 tested instances (Prob553, Prob557, Prob653, Prob654, Prob657, Prob754, Prob757, Prob758, Prob853, Prob854, Prob858), compared with the best obtained results of the other algorithms. The remaining results are either equal or close to other results.

From the experimental results mentioned in Tables 2 and 3, we can remark that GRASP-DE algorithm outperforms other algorithms for total tardiness criterion. We cannot compare the computational time efficiency because the authors used different hardwares and softwares and there are some authors who did not mention their computation times.

In Tables 4 and 5, we will focus on the presentation of the obtained results according to different values of parameter α. The results are presented in term of deviation, since we have various problem instances for which the optimum or the best known solution is 0. Therefore, we gathered each set of 8 problem instances in a class of jobs. The basic idea of the realization of this test is to show the contribution of the introduction of the learning process proposed according to time. The deviation of each algorithm is calculated by the following formula:

$$dev = \frac{(A - UB)}{UB} \times 100$$

where A denotes the average of each class achieved by each algorithm for a given problem and UB denotes the corresponding upper bound.

Tables 4 and 5 summarize our results. To check the efficiency of the proposed approach, we executed the constructive phase of our GRASP method, both with and without DE algorithm (denoted by GRASP-DE and GRASP respectively). To perform this comparative study, we used different values of the parameter α: $\alpha = 0.6$, $\alpha = 0.7$, $\alpha = 0.8$, $\alpha = 0.9$, $\alpha = 0.95$. We also varied and increased the number of iterations: 1000 iterations, 5000 iterations, 10000 iterations, 20000 iterations.

Doing so, we can prove the interest of using hybridization. The results show that the GRASP-DE algorithm performed better than GRASP on the problem instances in term of the deviation from the best known solution. Therefore, it is clear from Tables 4 and 5 that the GRASP-DE method is better than the GRASP method. For example,

Table 4 Performance comparison (constructive phase) of two algorithms GRASP & GRASP-DE for 1000 and 5000 iterations, when $\alpha = 0.6$, $\alpha = 0.7$, $\alpha = 0.8$, $\alpha = 0.9$, $\alpha = 0.95$

	1000 ($\alpha = 0.6$)						5000 ($\alpha = 0.6$)					
	GRASP			GRASP-DE			GRASP			GRASP-DE		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
15 jobs	9.47	14.31	18.06	6.59	12.40	17.53	8.75	12.39	15.60	2.16	6.02	8.93
25 jobs	36.16	43.40	49.16	26.87	40.65	53.61	30.84	38.25	43.54	18.87	24.82	31.74
35 jobs	26.39	32.04	35.52	29.68	37.12	43.82	24.90	29.81	32.95	22.88	27.59	32.72
45 jobs	31.49	35.08	38.18	38.05	45.99	53.66	27.66	32.45	35.55	28.91	33.81	37.81
55 jobs	42.62	46.89	50.18	53.64	60.47	67.58	40.73	44.44	47.34	40.56	45.13	49.81
65 jobs	37.35	41.99	45.28	50.08	57.34	63.96	36.65	39.99	42.39	39.46	43.81	47.83
75 jobs	42.24	47.46	50.43	59.49	68.78	76.46	41.17	44.80	47.46	44.49	49.22	54.00
85 jobs	34.48	37.57	39.43	46.29	51.65	57.79	32.19	35.46	37.42	37.27	40.20	45.54
AVG	32.53	37.34	40.78	38.84	46.80	54.30	30.36	34.70	37.78	29.32	33.83	38.55
	1000 ($\alpha = 0.7$)						5000 ($\alpha = 0.7$)					
	GRASP			GRASP-DE			GRASP			GRASP-DE		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
15 jobs	9.31	12.25	15.04	7.56	11.86	15.90	8.04	10.34	12.33	1.63	4.98	8.66
25 jobs	27.69	34.15	38.58	28.90	38.61	49.60	24.45	30.01	34.33	15.61	21.91	28.66
35 jobs	24.17	27.62	30.56	29.97	34.92	41.28	22.37	25.51	27.77	20.43	25.26	28.92
45 jobs	28.92	31.77	34.18	35.88	42.30	48.93	26.19	29.23	31.19	26.92	31.17	35.82
55 jobs	36.07	41.61	45.59	50.07	57.30	67.14	35.31	39.42	42.41	36.36	42.09	46.78
65 jobs	35.30	38.15	40.59	48.02	54.28	60.78	33.84	36.29	38.11	36.06	40.59	44.25
75 jobs	36.60	41.45	44.77	55.11	65.24	72.73	35.90	39.60	42.14	39.62	44.86	49.33
85 jobs	32.03	34.76	36.38	42.77	48.86	53.83	30.63	33.10	34.89	33.79	37.62	40.46
AVG	28.76	32.72	35.71	37.29	44.17	51.27	27.09	30.44	32.90	26.30	31.06	35.36
	1000 ($\alpha = 0.8$)						5000 ($\alpha = 0.8$)					
	GRASP			GRASP-DE			GRASP			GRASP-DE		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
15 jobs	11.52	13.10	14.45	5.53	10.53	15.51	10.44	12.10	13.22	1.32	3.95	7.23
25 jobs	22.00	27.31	31.51	23.71	35.08	48.89	19.63	24.17	27.69	14.38	19.45	25.35
35 jobs	20.19	22.86	25.05	26.17	33.47	39.59	18.18	21.16	22.82	18.81	23.24	26.96
45 jobs	25.01	27.84	30.23	35.01	40.80	46.75	23.25	25.97	27.91	23.90	27.89	33.36
55 jobs	32.25	34.81	37.12	45.90	54.35	61.00	29.33	32.87	35.00	34.01	38.78	43.66
65 jobs	31.27	33.89	35.59	45.66	52.16	57.73	29.60	32.32	34.03	32.70	36.68	40.19
75 jobs	32.20	34.82	36.86	53.07	61.12	68.89	30.34	33.21	35.02	35.03	39.96	44.44
85 jobs	29.91	31.67	33.16	40.87	46.54	51.50	28.55	30.40	31.67	30.63	34.21	37.13
AVG	25.54	28.29	30.50	34.49	41.76	48.73	23.67	26.52	28.42	23.85	28.02	32.29

we fixed the number of iterations at 20000. For $\alpha = 0.6$, we obtained 32.57% for GRASP and 27.50% for GRASP-DE. For $\alpha = 0.7$, we obtained 28.66% for GRASP

Table 4 (*Continued*)

	1000 ($\alpha = 0.9$)						5000 ($\alpha = 0.9$)					
	GRASP			GRASP-DE			GRASP			GRASP-DE		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
15 jobs	13.47	13.86	14.53	4.32	9.97	15.08	13.47	13.59	13.86	0.67	3.48	7.01
25 jobs	21.76	24.73	26.86	25.24	35.08	45.67	21.47	23.37	24.75	12.84	18.01	22.45
35 jobs	17.75	19.47	20.94	26.15	31.46	37.63	16.29	18.18	19.50	16.79	20.76	24.48
45 jobs	20.40	22.46	24.06	33.34	38.74	43.26	19.38	21.21	22.59	20.47	25.32	29.70
55 jobs	23.61	26.16	27.99	44.67	52.16	59.68	22.70	24.94	26.60	28.97	33.75	37.98
65 jobs	25.42	28.00	29.54	42.00	49.54	55.47	25.06	27.05	28.31	28.32	32.78	36.29
75 jobs	25.83	27.78	29.26	49.88	58.82	67.56	25.02	26.68	28.06	31.11	35.43	39.61
85 jobs	26.29	27.88	28.97	39.75	44.90	50.90	25.63	26.87	27.87	27.94	30.93	33.99
AVG	21.82	23.79	25.27	33.17	40.08	46.91	21.13	22.73	23.94	20.89	25.06	28.94
	1000 ($\alpha = 0.95$)						5000 ($\alpha = 0.95$)					
	GRASP			GRASP-DE			GRASP			GRASP-DE		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
15 jobs	16.93	16.93	16.93	6.12	10.26	14.58	16.93	16.93	16.93	1.09	3.47	6.80
25 jobs	23.81	25.49	27.34	23.11	33.14	44.61	23.75	24.53	25.71	9.67	16.55	22.53
35 jobs	16.04	17.25	18.11	25.30	31.66	36.26	15.59	16.57	17.34	17.04	20.23	25.28
45 jobs	21.13	22.34	23.19	29.42	38.41	45.48	20.61	21.64	22.37	19.93	24.29	28.17
55 jobs	20.90	22.36	23.54	43.29	51.95	59.19	19.90	21.40	22.35	27.07	32.36	39.14
65 jobs	24.10	25.72	26.87	43.37	49.11	55.41	23.30	24.65	25.53	26.96	31.55	35.38
75 jobs	23.08	24.18	25.18	47.51	57.49	69.54	22.02	23.40	24.37	28.69	32.98	37.60
85 jobs	23.95	25.11	26.02	36.73	43.71	49.01	23.40	24.45	25.19	25.94	29.27	33.20
AVG	21.24	22.42	23.40	31.86	39.47	46.76	20.69	21.70	22.47	19.55	23.84	28.51

and 24.91% for GRASP-DE. For $\alpha = 0.8$, we obtained 25.25% for GRASP and 21.90% for GRASP-DE. For $\alpha = 0.9$, we obtained 21.88% for GRASP and 18.54% for GRASP-DE. For $\alpha = 0.95$, we obtained 21.19% for GRASP and 16.64% for GRASP-DE.

For a given $\alpha = 0.6$, we can see that the results obtained by the GRASP algorithm become better when the number of iterations increases (37.34% for 1000 iterations, 34.70% for 5000 iterations, 33.42% for 10000 iterations and 32.57% for 20000 iterations). The same remark can be made for the proposed GRASP-DE algorithm (46.80% for 1000 iterations, 33.83% for 5000 iterations, 30.03% for 10000 iterations and 27.50% for 20000 iterations).

For a fixed number of iterations (1000 iterations), we can observe that the results obtained by the GRASP algorithm become better when the parameter α increases (37.34% for $\alpha = 0.6$, 32.72% for $\alpha = 0.7$, 28.29% for $\alpha = 0.8$, 23.79% for $\alpha = 0.9$ and 22.42% for $\alpha = 0.95$). The same remark can be made for the proposed GRASP-DE algorithm (46.80% for $\alpha = 0.6$, 44.17% for $\alpha = 0.7$, 41.76% for $\alpha = 0.8$, 40.08% for $\alpha = 0.9$ and 39.47% for $\alpha = 0.95$).

Table 5 Performance comparison (constructive phase) of two algorithms GRASP & GRASP-DE for 10000 and 20000 iterations, when $\alpha = 0.6$, $\alpha = 0.7$, $\alpha = 0.8$, $\alpha = 0.9$, $\alpha = 0.95$

	10000 ($\alpha = 0.6$)						20000 ($\alpha = 0.6$)					
	GRASP			GRASP-DE			GRASP			GRASP-DE		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
15 jobs	7.26	10.90	13.80	1.58	4.16	7.27	7.10	10.20	12.57	0.52	2.29	5.15
25 jobs	29.47	35.80	40.52	14.52	19.77	24.38	27.73	33.94	38.85	11.72	17.51	22.74
35 jobs	24.08	28.66	31.37	21.03	24.39	27.50	24.06	27.99	30.73	19.18	22.16	25.26
45 jobs	28.02	31.37	33.84	27.21	30.92	34.14	27.52	30.86	32.69	24.33	28.06	31.32
55 jobs	37.91	42.90	45.92	34.32	40.93	46.08	37.49	42.17	45.23	34.06	38.33	43.14
65 jobs	34.78	39.04	41.29	34.84	39.27	43.37	34.74	38.06	40.86	31.63	36.42	39.74
75 jobs	40.02	43.67	46.06	40.55	44.38	47.48	39.00	42.93	45.32	38.33	41.12	44.37
85 jobs	32.49	35.01	36.68	32.18	36.40	39.38	31.64	34.39	36.07	30.89	34.11	36.75
AVG	29.26	33.42	36.18	25.78	30.03	33.70	28.66	32.57	35.29	23.83	27.50	31.06
	10000 ($\alpha = 0.7$)						20000 ($\alpha = 0.7$)					
	GRASP			GRASP-DE			GRASP			GRASP-DE		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
15 jobs	7.96	9.72	11.73	1.24	3.10	6.86	7.52	9.20	10.71	0.20	1.94	4.47
25 jobs	20.48	28.29	33.63	13.33	17.59	21.96	23.03	27.08	31.06	10.46	14.45	18.37
35 jobs	20.83	24.17	26.75	17.85	22.09	26.00	20.22	23.31	25.29	16.46	19.97	22.98
45 jobs	25.62	28.30	30.54	24.07	27.71	31.74	24.62	27.60	29.81	20.27	25.08	29.43
55 jobs	33.21	38.25	40.82	33.76	37.89	41.84	34.15	37.37	39.68	30.62	35.35	38.82
65 jobs	32.38	35.10	37.20	31.72	36.37	40.30	30.89	34.62	36.71	29.78	33.52	37.53
75 jobs	36.13	38.70	40.58	36.00	40.53	44.51	36.10	38.08	39.90	33.51	37.28	40.74
85 jobs	30.46	32.33	33.74	30.67	34.36	37.62	29.98	32.02	33.24	28.77	31.65	34.32
AVG	25.88	29.36	31.88	23.58	27.45	31.35	25.81	28.66	30.80	21.26	24.91	28.33
	10000 ($\alpha = 0.8$)						20000 ($\alpha = 0.8$)					
	GRASP			GRASP-DE			GRASP			GRASP-DE		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
15 jobs	10.23	11.91	12.85	0.65	2.24	5.19	10.04	11.36	12.50	0.36	1.32	2.79
25 jobs	18.82	23.30	25.70	9.58	15.36	19.90	16.32	22.14	24.54	7.70	12.52	16.27
35 jobs	17.74	20.70	22.14	15.71	19.86	24.32	17.44	19.90	21.86	14.30	17.76	20.86
45 jobs	23.14	25.41	27.08	21.05	24.65	28.01	21.95	24.75	26.40	19.06	22.04	24.85
55 jobs	29.89	32.43	35.07	30.93	34.24	38.25	29.18	31.54	33.42	26.58	30.65	34.44
65 jobs	29.41	31.53	33.16	29.17	32.56	36.22	28.56	30.93	32.28	26.37	29.93	33.28
75 jobs	30.42	32.78	34.48	31.16	35.40	38.62	29.75	32.02	33.52	29.52	32.31	35.80
85 jobs	28.23	29.89	31.12	28.28	30.92	33.36	27.94	29.33	30.43	25.55	28.64	31.17
AVG	23.49	25.99	27.70	20.82	24.40	27.98	22.65	25.25	26.87	18.68	21.90	24.93

From Tables 4 and 5, we can observe an improvement of results obtained by GRASP-DE algorithm while the number of iterations increases. This proves the effectiveness of the learning process mechanism.

Table 5 (*Continued*)

	10000 ($\alpha = 0.9$)						20000 ($\alpha = 0.9$)					
	GRASP			GRASP-DE			GRASP			GRASP-DE		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
15 jobs	13.47	13.49	13.67	0.36	1.83	3.49	13.47	13.48	13.58	0.36	1.42	2.77
25 jobs	20.67	22.62	24.28	7.76	12.89	17.24	20.68	22.15	23.56	4.32	9.62	14.37
35 jobs	15.98	17.72	18.79	13.53	17.83	21.42	15.60	17.39	18.36	11.00	14.99	18.18
45 jobs	19.35	20.79	21.92	17.93	21.42	25.07	18.78	20.25	21.59	15.48	18.84	22.40
55 jobs	22.91	24.51	25.93	24.01	29.64	34.29	22.38	23.98	25.06	21.41	25.80	29.47
65 jobs	24.99	26.55	27.67	25.42	29.03	32.46	24.80	26.13	27.30	22.43	25.64	28.66
75 jobs	24.21	26.06	27.31	25.52	30.23	33.53	23.82	25.47	26.79	23.64	27.23	30.83
85 jobs	25.15	26.55	27.58	24.65	27.40	29.86	25.00	26.20	27.23	22.01	24.74	27.05
AVG	20.84	22.29	23.39	17.40	21.28	24.67	20.57	21.88	22.93	15.08	18.54	21.72
	10000 ($\alpha = 0.95$)						20000 ($\alpha = 0.95$)					
	GRASP			GRASP-DE			GRASP			GRASP-DE		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
15 jobs	16.93	16.93	16.93	0.37	1.91	5.00	16.93	16.93	16.93	0.36	1.21	3.66
25 jobs	23.67	24.09	25.13	7.21	12.06	16.55	23.65	23.88	24.65	4.22	8.65	13.16
35 jobs	15.42	16.25	17.01	13.02	16.64	19.69	15.47	16.02	16.53	9.98	13.61	17.11
45 jobs	20.35	21.31	21.98	15.99	20.30	24.37	20.05	21.02	21.73	14.08	17.13	20.35
55 jobs	19.54	21.09	21.96	22.54	27.31	32.60	19.94	20.83	21.74	18.94	22.81	27.25
65 jobs	23.19	24.38	25.50	22.63	26.70	30.97	23.28	24.22	24.98	19.74	23.09	26.37
75 jobs	21.77	23.01	23.81	24.39	27.85	30.92	21.51	22.58	23.40	19.79	23.92	27.49
85 jobs	23.27	24.19	24.84	22.43	25.43	28.20	23.23	24.02	24.61	19.63	22.68	25.66
AVG	20.52	21.41	22.14	16.07	19.77	23.54	20.51	21.19	21.82	13.34	16.64	20.13

Two important results can be pointed out. First, we demonstrated that, as the number of iterations and the parameter α increase, the initial solution generated by GRASP-DE is better than the initial one provided by GRASP. Hence, the hybridization, as learning process for probability adjustment, is useful for the improvement of the solution quality which is used as initial solution, for the Local Search algorithm.

Second, the numerical experiments proved that the best parameter for the two algorithms is $\alpha = 0.95$. Furthermore, the results were unsatisfying for the GRASP-DE algorithm when the number of iterations was low (1000 or 5000 iterations). However, results were improved as the number of iterations was increased.

This result reveals that it will be interesting to apply the Local Search algorithm, when our algorithm provides the initial solutions of good quality. Besides, we also try to exploit the good solutions that can be generated in the beginning of the algorithm execution. This remark shows the interest of using the selection probability $pr_{w(i)j}$. According to the level of this probability, the Local Search will be applied on the best initial solutions generated.

8 Conclusion

In this paper, we presented the single machine scheduling problem with SDST, with the objective to minimize total tardiness. Since this problem is known as NP-hard, a heuristic algorithm was developed. In this paper, we proposed a new approach based on the hybridization of GRASP and DE algorithms, in order to introduce a learning phase in the classical GRASP method for solving the single machine scheduling problem with SDST. The proposed algorithm was first tested, for "small size" problems, on 32 benchmark problem instances and secondly it was tested, for "large size" problems, on 32 benchmark problem instances.

Our approach outperformed all other algorithms proposed in the literature. In fact, this study demonstrated that our algorithm is very competitive compared with the other algorithms. It was compared with numerous algorithms respecting the total tardiness criterion. GRASP-DE algorithm found 12 solutions better than those of all the competing algorithms.

Furthermore, the GRASP-DE approach appeared better for small and large size problems than GRASP procedure used in the literature. Hence, the used hybridization was conducted on the improvement of the GRASP method. Moreover, computational experiments showed that this hybridization of GRASP and DE methods is very efficient. Since this proposed approach is effective in solving NP-hard problem, it can be experienced on a wide variety of combinatorial optimization problems. In future work, we will apply our method in order to solve the single machine total weighted tardiness scheduling problem with SDST. In this problem, each job has a weight that indicates its importance compared to other jobs. The objective is to minimize the total weighted tardiness.

References

1. Bigras, L.P., Gamache, M., Savard, G.: The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. Discrete Optim. **5**, 685–699 (2008)
2. Choobineh, F.F., Mohebbi, E., Khoo, H.: A multi-objective tabu search for a single-machine scheduling problem with sequence-dependent setup times. Eur. J. Oper. Res. **175**, 318–337 (2006)
3. Das, S.R., Gupta, J.N.D., Khumawala, B.M.: A savings index heuristic algorithm for flowshop scheduling with sequence dependent set-up times. J. Oper. Res. Soc. **46**, 1365–1373 (1995)
4. Du, J., Leung, J.Y.T.: Minimizing total tardiness on one machine is NP-hard. Math. Oper. Res. **15**, 483–495 (1990)
5. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. J. Glob. Optim. **6**, 109–133 (1995)
6. Feo, T.A., Venkatraman, K., Bard, J.F.: A GRASP for a difficult single machine scheduling problem. Comput. Oper. Res. **18**, 635–643 (1991)
7. Feo, T.A., Sarathy, K., McGahan, J.: A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. Comput. Oper. Res. **23**, 881–895 (1996)
8. Festa, P., Pardalos, P.M., Resende, M.G.C., Ribeiro, C.C.: GRASP and VNS for Max-Cut. In: Extended Abstracts of the Fourth Metaheuristics International Conference, pp. 371–376, Porto, July (2001)
9. França, P.M., Gendreau, M., Laporte, G., Müller, F.M.: A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. Int. J. Prod. Econ. **43**, 79–89 (1996)
10. França, P.M., Mendes, A., Moscato, P.: A memetic algorithm for the total tardiness single machine scheduling problem. Eur. J. Oper. Res. **132**, 224–242 (2001)

11. Gagné, C., Price, W.L., Gravel, M.: Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. J. Oper. Res. Soc. **53**, 895–906 (2002)
12. Gagné, C., Gravel, M., Price, W.L.: Using metaheuristic compromise programming for the solution of multiple-objective scheduling problems. J. Oper. Res. Soc. **56**, 687–698 (2005)
13. Gravel, M., Price, W.L., Gagné, C.: Scheduling jobs in an alcan aluminium foundry using a genetic algorithm. Int. J. Prod. Res. **38**, 3031–3041 (2000)
14. Gupta, S.R., Smith, J.S.: Algorithms for single machine total tardiness scheduling with sequence dependent setups. Eur. J. Oper. Res. **175**, 722–739 (2006)
15. Hansen, P., Mladenović, N.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**, 449–467 (2001)
16. Liao, C.J., Yu, W.C.: Sequencing heuristics for dependent setups in a continuous process industry. Omega—Int. J. Manag. Sci. **24**, 649–659 (1996)
17. Liao, C.J., Chen, W.J.: Scheduling under machine breakdown in a continuous process industry. Comput. Oper. Res. **31**, 415–428 (2004)
18. Liao, C.J., Juan, H.C.: An ant colony optimization for ACO single-machine tardiness scheduling with sequence-dependent setups. Comput. Oper. Res. **34**, 1899–1909 (2007)
19. Luo, X., Chu, F.: A branch and bound algorithm of the single machine schedule with sequence dependent setup times for minimizing total tardiness. Appl. Math. Comput. **183**, 575–588 (2006)
20. Luo, X., Chu, C.: A branch-and-bound algorithm of the single machine schedule with sequence-dependent setup times for minimizing maximum tardiness. Eur. J. Oper. Res. **180**, 68–81 (2007)
21. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**, 1097–1100 (1997)
22. Ozgur, C.O., Brown, J.R.: A two-stage traveling salesman procedure for the single machine sequence-dependent scheduling problem. Omega—Int. J. Manag. Sci. **23**, 205–219 (1995)
23. Potts, C.N., Van Wassenhove, L.N.: A branch and bound algorithm for the total weighted tardiness problem. Oper. Res. **33**, 363–377 (1985)
24. Ragatz, G.L.: A branch-and-bound method for minimum tardiness sequencing on a single processor with sequence dependent setup times. In: Proceedings of the twenty-fourth annual meeting of the Decision Sciences Institute, pp. 1375–1377 (1993)
25. Rubin, P.A., Ragatz, G.L.: Scheduling in a sequence dependent setup environment with genetic search. Comput. Oper. Res. **22**, 85–99 (1995)
26. Selen, W.J., Heuts, R.M.J.: Operational production planning in a chemical manufacturing environment. Eur. J. Oper. Res. **45**, 38–46 (1990)
27. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**, 341–359 (1997)
28. Sun, X., Noble, J.S., Klein, C.M.: Single-machine scheduling with sequence dependent setup to minimize total weighted squared tardiness. IIE Trans. **31**, 113–124 (1999)
29. Tan, K.C., Narasimhan, R.: Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach. Omega—Int. J. Manag. Sci. **25**, 619–634 (1997)
30. Tan, K.C., Narasimhan, R., Rubin, P.A., Ragatz, G.L.: A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. Omega—Int. J. Manag. Sci. **28**, 313–326 (2000)
31. Ying, K.C., Lin, S.W., Huang, C.Y.: Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. Expert Syst. Appl. **36**, 7087–7092 (2009)