Int J Adv Manuf Technol (2009) 44:795–808
DOI 10.1007/s00170-008-1885-7

ORIGINAL ARTICLE

Due window scheduling with sequence-dependent setup on parallel machines using three hybrid metaheuristic algorithms

J. Behnamian · M. Zandieh · S. M. T. Fatemi Ghomi

Received: 25 June 2008 / Accepted: 28 November 2008 / Published online: 13 January 2009

Abstract Due-date determination problems have gained significant attention in recent years due to the industrial focus in the just-in-time philosophy. This paper considers a machine scheduling problem where jobs should be completed at times as close as possible to their respective due dates, and hence, both earliness and tardiness should be penalized. It is assumed that earliness and tardiness (ET) penalties will not occur if a job is completed within the due window. However, ET penalties will occur if a job is completed outside the due window. The objective is to determine a schedule that minimizes sum of the earliness and tardiness of jobs. To achieve this objective, three hybrid metaheuristics are proposed. The first metaheuristic is a hybrid algorithm which combines elements from both simulated annealing (SA) as constructive heuristic search and a variable neighborhood search (VNS) as local search improvement technique. The second one presents a hybrid metaheuristic algorithm which composed of a population generation method based on an ant colony optimization (ACO) and a VNS to improve the population. Finally, a hybrid metaheuristic approach is proposed which integrates several features from ACO, SA, and VNS in a new configurable scheduling algorithm. A design of experiments approach is employed to calibrate the parameters and operators of the algorithm. Computational experiments conducting on 252 randomly generated problems compare the results with the VNS algorithm proposed previously and show that the procedure is capable of producing consistently good results.

Keywords Parallel machines scheduling · Sequence-dependent setup times · Due window scheduling · Variable neighborhood search · Ant colony optimization · Simulated annealing

J. Behnamian · S. M. T. Fatemi Ghomi
Department of Industrial Engineering,
Amirkabir University of Technology,
424 Hafez Avenue,
Tehran, Iran

M. Zandieh (✉)
Department of Industrial Management,
Management and Accounting Faculty,
Shahid Beheshti University,
Tehran, Iran
e-mail: m_zandieh@sbu.ac.ir

1 Introduction

A useful but difficult criterion in scheduling theory is the minimization of both earliness and tardiness values. It comes from the "just-in-time" (JIT) philosophy in management and production theory: An item should be delivered exactly when it is required by the customer. Therefore, both early and tardy deliveries of a task with respect to its due date are penalized. The earliness and tardiness problem was originally called the minimum weighted absolute deviation problem, although since about 1990, it has been commonly referred to as the ET problem [2]. JIT scheduling models assume the existence of job due dates and discourage early as well as tardy jobs. If a job finishes before its due date, it incurs an earliness penalty such as holding cost. On the other hand, completing the job after its due date can lead to such tardiness costs as late charges, express delivery charges, or lost sales. A schedule that minimizes the sum of these penalties is said to be a JIT schedule. This paper assumes that earliness and tardiness are penalized at the same rate for all jobs.

In the last two decades, many results have appeared in the scheduling literature that consider both earliness and tardiness penalties. There are two related scheduling models

studied in the literature. In the first model, the due date of a job is only a point of time. There is no penalty if the job is completed exactly at this point of time. An earliness penalty is incurred if the job is completed before this time. More recent results on this model include, among others, Emmons [18], Kubiak et al. [36], Lee et al. [40], Herrmann and Lee [28], Cheng and Chen [16], Chen [13], Federgruen and Mosheiov [20], Almeida and Centeno [4], Chen and Powell [15], Birman and Mosheiov [10], Esteve et al. [19], and Bülbül et al. [11]. Recent surveys on these problems have been published by Hoogeveen [30], Lauff and Werner [39], and Gordon et al. [25].

The second model, which is more realistic, assumes that the due date of a job is an interval rather than a point in time. This interval of time is called the due window of the job, and the left end and the right end of the window are called the earliest due date and the latest due date of the job, respectively. No penalty is incurred if the job is completed at any time within its due window. An earliness penalty is incurred if the job is completed before its earliest due date. Clearly, this model is more general and practical than the first one since it includes the first model as its special case with due window of size zero. On the other hand, problems with this model are generally more difficult to solve than those with the first model. The earliest work on this model was by Anger et al. [5] who considered the small size job due window, and the objective is to minimize the number of jobs that do not finish within their respective due windows. Since then, several papers have appeared to address JIT scheduling problems with due windows under different criteria. The majority of the literature on JIT scheduling deals with only single machine scheduling problems. The work of Kramer and Lee [35], Chen and Lee [14], Wan and Yen [59], and Yeung et al. [61] are a few papers that study the multifacility problem with the due window model, and other multifacility problems consider job due date at a point of time.

In literature, just a few authors consider scheduling problem with sequence-dependent setup times. Heady and Zhu [27] minimized the total weighted deviation of the completion times from the given due dates, where machines may be unrelated and sequence-dependent setup times are given. Lam and Xing [38] gave a short review of new developments of parallel machine scheduling problem (PMSP) associated with the problems of just-in-time productions, preemption with setup and capacitated machine scheduling. Balakrishnan et al. [7] used a mixed integer formulation that had substantially fewer zero one variables than typical formulations for scheduling problems of this type. The parallel machine scheduling problem of minimizing total earliness and tardiness with a common weight for earliness and tardiness, respectively, as well as given sequence-dependent setup times for the jobs was treated by Sivrikaya-Serifoglu and Ulusoy [51]. Machines are either identical or uniform. For this problem, two genetic algorithms (GAs) were given and tested on problems with up to 60 jobs and four machines. With respect to minimizing the total weighted tardiness in unrelated parallel machines scheduling problem, Vignier et al. [58] proposed a hybrid method that consists of an iterative heuristic, a genetic algorithm, and a branch-and-bound algorithm to solve sequence-dependent setup times parallel machine scheduling problem with nonzero release date, where there are two types of machines, both processing and setup times depend on the machines, and each job has a release date and a due date.

Radhakrishnan and Ventura [48] consider the case of sequence-dependent setup time parallel machine scheduling problem with the objective of minimizing the sum of the total earliness and total tardiness. To solve the problem, they propose a mathematical programming formulation that can be used for limited-sized problems and proposed a simulated annealing algorithm for large-sized problems.

Regarding uniform parallel machines and sequence-dependent setup times, we can refer Tamimi and Rajan [55]. They proposed a genetic algorithm to minimize the total weighted tardiness. In their genetic algorithm, they dynamically modified the mutation rate, crossover rate, and insertion rate. For the same objective, Park et al. [46] proposed the use of a neural network to obtain values for the parameters in calculating a priority rule for the sequence-dependent setup time parallel machine scheduling. Their computational results indicated that the proposed approach outperforms that of an earlier approach.

Kurz and Askin [37] used an integer programming formulation for sequence-dependent setup time parallel machine scheduling with nonzero release date. They developed several heuristic methods including a genetic algorithm and multifit-based approaches and empirically evaluated them. Gendreau et al. [24] proposed lower bounds and a divide and merge heuristics for addressed problem. They compared their heuristic method with earlier heuristic methods of tabu search and showed that their heuristic is much faster while producing similar quality results. Kim et al. [32] attempted to solve the unrelated parallel machines and sequence-dependent setup times through a new approach. In this particular case, the setup times are not machine dependent and each job is composed of n items. In the line of batch scheduling problems, we can cite Kima et al. [33], where several search heuristics are presented and tested considering unrelated parallel machines.

Hiraishi et al. [29] investigated identical parallel machine scheduling with sequence-dependent setup times to maximize the weighted number of jobs completed at their due dates. Fowler et al. [22] develop a hybrid genetic algorithm

for the sequence-dependent setup time parallel machine scheduling with nonzero release date. They considered three separate problems with objective of makespan, total weighted completion time, and total weighted tardiness. In their algorithm, a genetic algorithm is used to assign jobs to machines, and dispatching rules are used to schedule the individual machines. Applying tabu search into sequence-dependent setup time parallel machine scheduling with nonzero release date was reported by Bilge et al. [9]. They investigated several key components of tabu search and identified the best values for these components.

Feng and Lau [21] use metaheuristic called squeaky wheel optimization to schedule jobs on more general problem with the objective of minimizing the sum of the total weighted earliness and total weighted tardiness. They showed that their heuristic method outperforms that of Radhakrishnan and Ventura [48]. The sequence-dependent setup time parallel machine scheduling with nonzero release date was addressed by Nessah et al. [45]. For the total weighted completion time minimization, they proposed a heuristic method based on the defined conditions. They also developed a lower bound. The quality of their heuristic method was tested on randomly generated problems by comparing the heuristic solution with a developed lower bound. Tahar et al. [53] presented a heuristic method based on linear programming modeling to minimize maximum completion time for sequence-dependent setup time parallel machine scheduling. The performance of their proposed method was tested on problems with a lower bound.

Logendrana et al. [42] deal with the minimization of the weighted tardiness of jobs in unrelated parallel machines scheduling with sequence-dependent setups considering dynamic release of jobs and dynamic availability of machines. Rocha et al. [50] proposed a variable neighborhood search algorithm to minimize the sum of the completion times plus the sum of the weighted delays. They showed that their algorithm outperforms a greedy randomized adaptive search procedure algorithm.

This paper removes the due date assumption and replaces it with a due window assumption so as to minimize the sum of ET penalties. Note that our problem is a real-life one, since most of the studies on parallel machine scheduling problem setup times are not taken into account to minimize the sum of the earliness and tardiness. The setup time has often been considered to be negligible or as a part of the processing time.

The scheduling problem with parallel machines and sequence-dependent setup times is known to be of difficult solution. The single-machine scheduling problem with sequence-dependent setups is known to be nondeterministic polynomial (NP)-hard [47]. And for the parallel machine case, it is proved that the problem of minimizing the makespan with two identical machines is NP-hard [41]. Thus, the more complex case of minimizing sum of the ET penalties on a scheduling problem with m identical machines is also NP-hard. Therefore, the use of metaheuristics is appropriate. In this paper, a hybrid metaheuristic (HMH) customizable approach is proposed, which allows the definition of scheduling algorithms by appropriately selecting and combining several different features derived from three main metaheuristics of reference, i.e., the ant colony optimization, the simulated annealing, and the variable neighborhood search. The purpose is to evaluate the effectiveness of integrating such a specific subset of features into a configurable hybrid metaheuristic, at least for the class of difficult scheduling problems here considered, consisting of the generalization of the PMSP.

The paper is organized as follows. Section 2 is the problem description. Section 3 introduces the proposed hybrid algorithm. Section 4 presents the two VNS-based hybrid algorithms used in the experiments. Section 5 presents and compares the hybrid algorithm with two versions of VNS-based hybrid algorithm. Finally, Section 6 states our conclusions and further researches on this topic.

2 Problem description

This paper studies the problem of scheduling a set of n independent jobs on m identical parallel machines. Each job j has a processing time p_j. When a job k is processed after job j, a setup time s_{jk} is incurred. All the jobs have a due window $[d_{j1}, d_{j2}]$, where d_{j1} is the earliest due date and d_{j2} ($d_{j2} \geq d_{j1}$) the latest due date. Furthermore, we assume that this due window is given in advance, i.e., both d_{j1} and d_{j2} are prespecified. Each job $j \in n$ is ready at time 0 and has a known processing time p_j. If the job $j \in n$ is completed before d_{j1}, it will incur an earliness penalty and if the job is completed after d_{j2}, it will incur tardiness penalty.

3 The hybrid metaheuristic approach

The interest in the design and implementation of hybrid metaheuristics has increased remarkably in recent years [54]. However, to date, there has been little research effort dedicated to the hybridization of metaheuristics for scheduling problem. The construction of hybrid metaheuristics is motivated by the need to achieve a good tradeoff between the capabilities of a heuristic to explore the search space and the possibility to exploit the experience accumulated during the search. In this work, the proposed hybrid metaheuristic, integrating features also from different approaches, should not be considered as an attempt to transform a "main" metaheuristic (e.g., say the ant colony optimization) into a hybrid metaheuristic by grafting some

contributions from other metaheuristics. Differently, the HMH allows the possible integration of several basic features from a reference set of metaheuristics and aims at analyzing the effectiveness of the resultant customizable algorithm for a difficult problem and a test problem.

3.1 Encoding scheme

The first important step in introduction of HMH algorithm to parallel machine problems is the representation of a solution. One of the most important decisions in designing a metaheuristic lies in deciding how to represent solutions and relate them in an efficient way to the searching space. Bean [8] introduced a random key approach for real-coded GA to solve sequencing problem. Subsequently, numerous researchers show that this approach is robust and can be applied for the solution of different kinds of scheduling problems [12, 56]. In this approach, random numbers serve as sort keys in order to decode the solution. The decoded solution is evaluated with a fitness function that is appropriate for the problem at hand. Each job is assigned a real number whose integer part is the machine number to which the job is assigned and whose fractional part is used to sort the jobs assigned to each machine. In this paper, to represent a schedule incorporated with the hybrid methodology, the random key approach is applied to address problem. Let us now discuss which aspects have been borrowed from ACO, VNS, and SA.

3.2 Ant colony optimization

Ant colony optimization is a metaheuristic to tackle hard combinatorial optimization (CO) problem that was first proposed in the early 1990s [17]. The inspiring source of ACO is the foraging behavior of real ants. Real ants have the ability to find the shortest path from a food source to their nest without using visual cues. Instead, they communicate information about the food source by producing a chemical substance, called pheromone, laid on their paths. Since the shorter paths have a higher traffic density, these paths can accumulate a higher amount of pheromone. Hence, the probability of ants following these shorter paths will be higher than the longer ones. The ACO algorithm implemented in this paper is basically the ant colony system version, but its pheromone values are limited to a bounded value. Explicit limits on the pheromone values prevent the probability to construct a solution falls below a certain value greater than 0. This algorithm is called max–min ant system (MMAS) [52]. The complete details of the algorithm are presented in view of the fact that ACO algorithms are relatively less knowing, as against simulated annealing or genetic algorithms, with respect to the application to scheduling problems.

The following presents the basic steps of the MMAS:

Step 1: Initialize the pheromone trails and parameters.
Step 2: While termination condition is not met, do the following:

- •Construct a solution.
- •Improve the solution by local search.
- •Update the pheromone trail or trail intensity, denoted by τ_{ip}, where $\tau_{\max} \geq \tau_{ip} \geq \tau_{\min}$.

Step 3: Return the best solution found.

In the context of application of the ACO algorithm to scheduling problems, τ_{ip} denotes the trail intensity (or desire) of setting job i in position p of a sequence. These trails form a kind of adaptive memory of previously found solutions and are modified at the end of every iteration. It is to be noted that, for every job i, for any possible position p, a pheromone value is stored and updated in each iteration of the ACO algorithm. Hence, there are n^2 such values of τ_{ip}. In order to ensure that the trail intensities do not go beyond certain limits during the process of updating the trail, Stützle [52] introduced the maximum and minimum values for τ_{ip}s, denoted by $\tau_{\max}$ and $\tau_{\min}$, respectively. This more applicable ACO algorithm is shown in Algorithm 1.

Algorithm 1: Ant Colony Optimization

```
1:  input: an instance x of a CO problem
2:  while termination conditions not met do
3:  Schedule activities
4:  Ant based solution construction()
5:  Pheromone update()
6:  Daemon actions()
7:  end schedule activities
8:  S_best ← best solution in the population of solutions
9:  end while
10: output: S_best, "candidate" to optimal solution for x
```

3.2.1 Generating the initial solution with an ant sequence

Initially, all τ_{ip}s are set equal to $\tau_{\max}$, where $\tau_{\max}$ is set equal to $1/((1-\rho)\times Z_{\text{best}})$ and ρ denotes the persistence of the trail. The term "Z_{best}" refers to the best value of objective function obtained so far. Initially, Z_{best} equals the objective function value yielded by the seed sequence. Starting from a null sequence, ant colony optimization algorithm makes use of trail intensities to determine the job to be appended in position p, where $1 \leq p \leq n$ and n refers to the number of jobs to be scheduled. Although the trail intensity τ_{ip} changes in every iteration of ant colony optimization algorithm, the iteration counter is omitted simplicity for simplicity of presentation. An ant starts constructing a sequence by choosing a job for the first position, followed

by the choice of an unscheduled job for the second position, and so on. A dummy job "0" is introduced on which an ant is set initially, and the construction of partial sequences begins, thereby leading to the buildup of a complete sequence by the ant, called an ant sequence. In the case of MMAS, the following procedure is used to choose an unscheduled job, say job i, probabilistically for position p.

Choose randomly a machine j
Sample a uniform random number u in the range (0,1).
If $u \leq (n-4)/n$
then

among the jobs that are not yet scheduled, choose the job with the maximum value of τ_{ip} to assign on machine j;

else

job i is selected, from the set of first five unscheduled jobs as present in the best sequence obtained so far, for position p by sampling from the following probability distribution:

$$\varepsilon_{ip} = \left(\tau_{ip} \Big/ \sum_{i''} \tau_{i''p} \right), \tag{1}$$

where job i'' belongs to the set of first five unscheduled jobs, as present in the best sequence obtained so far. Note that when there are less than five jobs unscheduled, all such unscheduled jobs are considered.

3.2.2 Updating the trail intensities

An ant sequence is subjected to the position-based local search procedure to enhance the quality of solution. Let the objective function value of this improved sequence be denoted by Z_{current}. The trail intensities are updated as follows:

$$\tau_{ip}^{\text{new}} = \begin{cases} \rho \times \tau_{ip}^{\text{old}} + (1/Z_{\text{current}}) & \text{if job } i \text{ is placed in position } p \text{ in the generated sequence;} \\ \rho \times \tau_{ip}^{\text{old}} & \text{otherwise,} \end{cases} \tag{2}$$

Update the best sequence and Z_{best}, if the generated sequence is superior to the best sequence obtained so far [23].

3.3 Variable neighborhood search

VNS is one of the most recent metaheuristics developed for problem solving in an easier way. It is known as one of the very well-known local search methods [43], takes more attention day by day, because of its ease of use and success in solving combinatorial optimization problems [26]. Basically, a local search algorithm carries out exploration within a limited region of the whole search space. This only facilitates to find better solutions without doing further investigation. The VNS is a simple and effective search procedure that proceeds to a systematic change of neighborhood. An ordinary VNS algorithm starts with an initial solution, $x \in S$, where S is the whole set of search space, and manipulates it through a two-nested loop in which the core one alters and explores via two main functions so-called shake and local search. The outer loop works as a refresher reiterating the inner loop, while the inner loop carries out the major search. Local search explores for an improved solution within the local neighborhood, while shake diversifies the solution by switching to another local neighborhood. The inner loop iterates as long as it keeps improving the solutions, where an integer, k, controls the length of the loop. Once an inner loop is completed, the outer loop reiterates until the termination condition is met. Since the complementariness of neighborhood functions is the key idea behind VNS, the neighborhood search (NS) should be chosen very rigorously so as to achieve an efficient VNS.

This method is different from the most local search heuristics in that it uses two or more neighborhoods, instead of one, in its structure. In addition, to avoid costing too much computational time, the best number of neighborhoods is often three [50], which is followed by our algorithm, so index l is defined to show local search type.

Note that the neighborhoods themselves ($N_1(S)$, $N_2(S)$, and $N_3(S)$, respectively) are determined both by its respective structure and by the solution it is being applied to. The size of neighborhood $N_1(S)$ is $O(m \cdot n^2)$, neighborhood $N_2(S)$ is $O(m^2 \cdot n^2)$ and neighborhood $N_3(S)$ is $O(n^2)$. The basic VNS structure is designed as shown in Algorithm 2.

Algorithm 2: Basic VNS structure

1: Find an initial solution S^*;
2: $l \leftarrow 1$;
3: **for** *iterations* $\leftarrow$ 1 to a maximum number of *iterations* **do**
4: $S \leftarrow S^*$;
5: Shake procedure: find a random solution $S' \in N_l(S)$;
6: Perform a local search on $N_l(S')$ to find a solution S'';
7: **if** $f(S'') \leq f(S^*)$ **then**
8: $S^* \leftarrow S''$;
9: $l \leftarrow 1$;
10: **end if**
11: $l \leftarrow l+1$;
12: **end for**

3.3.1 Random solutions

Every time a neighborhood is selected, a random procedure is called. This procedure selects a random solution from the selected neighborhood structure. Therefore, three procedures are created in the following manner, one for each index l:

1. For $N_1(S)$:
   - Choose randomly a machine i.
   - Choose randomly two jobs j_1 and j_2 on machine i.
   - Swap jobs j_1 and j_2.
2. For $N_2(S)$:
   - Choose randomly two machines i_1 and i_2.
   - Choose randomly a job j_1 on i_1 and a job j_2 on i_2.
   - Swap jobs j_1 and j_2.
3. For $N_3(S)$:
   - Choose randomly one job j_1 and one machine i_2, where j_1 does not belong to i_2.
   - Choose randomly a valid position "*pos*" in i_2.
   - Transfer job j_1 to i_2 at the position *pos*.

3.3.2 The local searches

There are several variations of the VNS structure. In our version, we use a specific local search for each neighborhood. The local searches are listed below.

LS 1: Job swaps at one machine: This local search analyzes every possible swap on one machine (see Fig. 1). Even when the chosen machine is not the one with the greatest completion time, the objective function can be improved by reducing the delay of some jobs. The algorithm has the time complexity $O(m.n^2)$:

Algorithm 3: Local search 1

1: **for each** i **do**
2: **for each** j_1 in i **do**
3: **for each** j_2 in i, $j_1 \neq j_2$, **do**
4: **if** solution considering j_1 and j_2 swapped < current solution **then**

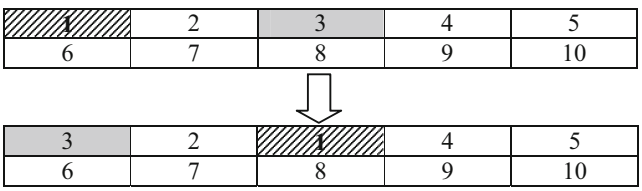

Fig. 1 Local search 1

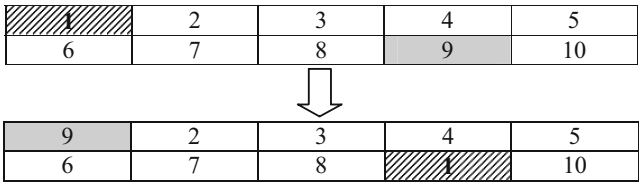

Fig. 2 Local search 2

5: Swap j_1 and j_2.
6: **end if**
7: **end if**
8: **end for**
9: **end for**

LS 2: Job swaps on different machines: In this local search, all job swaps between jobs belonging to these different machines are evaluated (see Fig. 2). A larger number of solutions are searched. The time complexity of our algorithm is $O(m^2.n^2)$:

Algorithm 4: Local search 2

1: **for each** i_1 **do**
2: **for each** j_1 in i_1 **do**
3: **for each** i_2, $i_1 \neq i_2$, **do**
4: **for each** $j_2 \in i_2$ **do**
5: **if** solution considering j_1 and j_2 swapped < current solution **then**
6: Swap j_1 and j_2.
7: **end if**
8: **end for**
9: **end for**
10: **end for**
11: **end for**

LS 3: Job insertion: This procedure searches for new solutions transferring jobs from the machine with the highest sum of the earliness and tardiness to the machine with the lowest one (see Fig. 3). The time complexity of our implementation is $O(n^2)$:

Algorithm 5: Local search 3

1: Find the machine with the highest sum of the earliness and tardiness i_1;
2: Find the machine with the lowest sum of the earliness and tardiness i_2, $i_1 \neq i_2$;
3: **for each** j in i_1 **do**

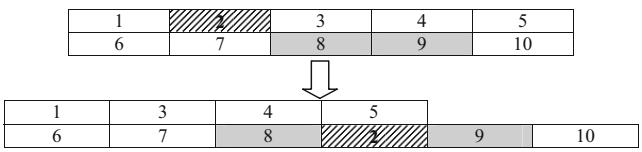

Fig. 3 Local search 3

4: **for each** valid position *pos* in i_2 **do**
5: **if** solution considering *j* transferred from i_1 to i_2 in position *pos* < current solution **then**
6: Transfer *j* from i_1 to i_2 on position *pos*.
7: **end if**
8: **end for**
9: **end for**

The algorithm always tries to use the fastest local search available first. If after an iteration no improvement is made, another neighborhood is used (l is incremented), and every time a new solution is found, the first and the fastest local search is used (l=1). Further reading about neighborhood search techniques can be found on Ahuja et al. [3].

3.4 Simulated annealing

The SA technique, proposed by Kirkpatrick et al. [34], is an iterative, stochastic, neighborhood-based search method motivated from an analogy between the simulation of the annealing of solids and the strategy of solving combinatorial optimization problems. SA has been widely applied to solve combinatorial optimization problems [60]. It is inspired by the physical process of heating a substance and then cooling it slowly, until a strong crystalline structure is obtained. This process is simulated by lowering an initial temperature by slow stages until the system reaches to an equilibrium point, and no more changes occur. The algorithmic framework of SA is described in Algorithm 6.

Algorithm 6: Simulated annealing

1: **input**: an instance *x* of a CO problem
2: $S \leftarrow$ Generate Initial solution()
3: $k \leftarrow 0$
4: $T_k \leftarrow$ Set Initial temperature()
5: **while** termination conditions not met **do**
6: $S' \leftarrow$ Pick neighbor at random($N(S)$)
7: **if** $f(S') \leq f(S)$ **then**
8: $S \leftarrow S'$;
9: **else**
10: Accept S' as new solution with probability $p(T_k, S', S)$
11: **end if**
12: Adapt temperature(T_k)
13: **end while**
14: $S_{\text{best}} \leftarrow S$
15: **output**: S_{best}, "candidate" to optimal solution for *x*

The main idea of this technique is to start from some initial solution, π', and successively move among neighboring solutions until the stopping condition is satisfied. At each iteration, *iter*, a random solution, π', is selected from the neighborhood of actual solution (π_{iter}) and it replaces with a probability

$$P(t_{\text{iter}}, \pi_{\text{iter}}, \pi') = \min\left\{1, \exp\left(-\frac{f(\pi') - f(\pi_{\text{iter}})}{t_{\text{iter}}}\right)\right\} \tag{3}$$

where t_{iter} is a parameter called the temperature at iteration *iter*. The temperature decreases during the search process according to the cooling scheme. The performance of SA depends on the following parameters, which have to be precisely selected:

- Initial temperature
- Cooling scheme
- Final temperature

The following presents the implementation of SA algorithm. For detailed description of the SA method, the reader is referred to the literature, see, e.g., Aarts and Lenstra [1], Kirkpatrick et al. [34], and Tian et al. [57].

3.4.1 Initial temperature

The initial temperature is selected on the basis of $K=nm+1$ solutions π_0; π_1;...; π_K, where π_j is randomly selected from the neighborhood of π_{j-1} and π_0 is an initial solution. The initial temperature is defined as

$$t_0 = \varphi_1 \frac{\delta}{mn}, \tag{4}$$

where $\delta = \max_{j=1,\ldots,k}\{f(\pi_j) - f(\pi_{j-1})\}$.

3.4.2 Cooling scheme

The temperature changes in every iteration according to the logarithmic cooling scheme:

$$t_{j+1} = \frac{t_j}{1 + \lambda t_j}, \tag{5}$$

where parameter λ is defined as

$$\lambda = \frac{t_0 - t_f}{f t_f t_0} \tag{6}$$

and f is 200. The final temperature t_f is determined from the following expression [31]:

$$t_f = \varphi_2 \frac{f(\pi_0)}{mn}. \tag{7}$$

3.5 Hybrid algorithm

The hybrid algorithm proposed here combines three methods previously presented. The steps of the HMH algorithm are as follows:

Step 1: Generating the initial solution with an ant sequence. The method starts with generating initial solutions by employing ACO.

Step 2: Intensification phase using VNS. The concept borrowed by the HMH from the VNS is that varying the neighborhood structure during the search process could facilitate the avoidance of traps and enlarge the search scope. In this first phase, the ET value for each neighborhood solution is calculated. VNS works by performing movements that upgrade the solutions. So, the solution that has the minimal ET value is selected as a move.

Step 3: Diversification phase via a SA and VNS. In HMH, a SA only is applied to diversification the solution. The asymptotic convergence of SA to a global optimum has been proved, even if such a result has only a theoretical relevance [6]. At iteration, a solution π' is selected from the neighborhood based on the VNS technique. When each movement is available, unlike VNS technique, SA executes a second phase where moving from the current solution to a worse solution is allowed with the probability given by expression Eq. 3, with the expectation that this movement will eventually guide to a better solution. However, if π' is not accepted, neighborhood structure is changed in the same way as in VNS technique (shaking function).

Step 4: Global pheromone update phase and ant generation for next solution by means of ACO. This phase is performed after each ant has completed its schedule. In order to make the search more directed, the global pheromone update rule is intended to better schedules.

The proposed hybrid metaheuristic mechanics is schematically presented in Fig. 4. The basic proposed hybrid algorithm structure is designed as shown in Algorithm 7.

Algorithm 7: Main body of hybrid algorithm (namely HMH)

1: $R_{\text{time}} \leftarrow$ Set run time()
2: **while** run time $< R_{\text{time}}$ **do**
3: $S^* \leftarrow$ Generate initial solution()
4: $l \leftarrow 1$;
5: **for** each ant **do**
6: $S \leftarrow S^*$;
7: $k \leftarrow 0$
8: $T_k \leftarrow$ Set initial temperature()
9: $T_f \leftarrow$ Set final temperature()
10: **while** *current temperature* $> T_f$ **do**
11: Shake procedure: find a random solution $S' \in N_l(S)$;
12: Perform a local search on $N_l(S')$ to find a solution S'';
13: **if** $f(S'') \leq f(S^*)$ **then**
14: $S^* \leftarrow S''$;
15: $l \leftarrow 1$;
16: **else**
17: Accept S'' as new solution with probability $p(T_k, S'', S^*)$
18: **end if**
19: Adapt temperature(T_k)
20: $l \leftarrow l+1$;
21: **end while**
22: generate schedule
23: evaluate schedule
24: **end for**
25: verify for global or local best
26: evaporate pheromone in all trials
27: deposit pheromone on best global schedule
28: **end while**

4 VNS-based hybrid algorithms

This section discusses in details the proposed two VNS-based algorithms with reference to the following steps that summarize the hybridization procedure. In each algorithm, first we generate the initial solution randomly then combine the main metaheuristic ACO and SA algorithm with variable neighborhood search. Basically, these hybrid metaheuristics consist of two parts: the construction of a main algorithm and a local search that make with VNS algorithm. Details of metaheuristic used in these algorithms are same as the hybrid approach that we describe

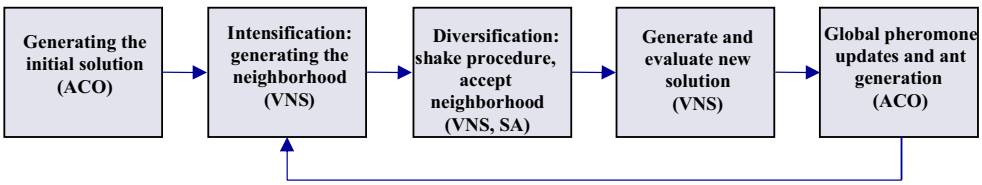

Fig. 4 The hybrid metaheuristic mechanism step by step

previously. The algorithmic frameworks of two VNS-based hybrid metaheuristics are described in Algorithms 8 and 9.

Algorithm 8: Main body of hybrid ACO/VNS (namely ANV)

1: $R_{\text{time}} \leftarrow$ Set run time()
2: **while** run time < R_{time} **do**
3: $S^* \leftarrow$ Generate initial solution()
4: $l \leftarrow 1$;
5: **for** each ant **do**
6: $S \leftarrow S^*$;
7: $k \leftarrow 0$
8: **for** stopping criteria meet **do**
9: Shake procedure: find a random solution $S' \in N_l(S)$;
10: Perform a local search on $N_l(S')$ to find a solution S'';
11: **if** $f(S'') \leq f(S^*)$ **then**
12: $S^* \leftarrow S''$;
13: $l \leftarrow 1$;
14: **end if**
15: Adapt temperature(T_k)
16: $l \leftarrow l+1$;
17: **end for**
18: Generate schedule
19: Evaluate schedule
20: **end for**
21: Verify for global or local best
22: Evaporate pheromone in all trials
23: Deposit pheromone on best global schedule
24: **end while**

Algorithm 9: Main body of hybrid SA/VNS (namely SAV)

1: $R_{\text{time}} \leftarrow$ Set run time()
2: **while** run time < R_{time} **do**
3: $S^* \leftarrow$ Generate initial solution()
4: $l \leftarrow 1$;
5: **for** *iterations* $\leftarrow$ 1 to a maximum permitted number of *iterations* **do**
6: $S \leftarrow S^*$;
7: $k \leftarrow 0$
8: $T_k \leftarrow$ Set initial temperature()
9: $T_f \leftarrow$ Set final temperature()
10: **while** *current temperature* > T_f **do**
11: Shake procedure: find a random solution $S' \in N_l(S)$;
12: Perform a local search on $N_l(S')$ to find a solution S'';
13: **if** $f(S'') \leq f(S^*)$ **then**
14: $S^* \leftarrow S''$;
15: $l \leftarrow 1$;
16: **else**
17: Accept S'' as new solution with probability $p(T_k, S'', S^*)$
18: **end if**
19: Adapt temperature(T_k)
20: $l \leftarrow l+1$;
21: **end while**;
22: **end for**
23: Verify for global or local best
24: **end while**

5 Experimental design

5.1 Data generation and settings

An experiment was conducted to test the performance of the hybrid algorithms. Following Kurz and Askin [37], data required for a problem consist of the number of jobs, number of machines, range of processing times, and the range of sequence-dependent setup times. The ready times are set to 0 for all jobs. To analyze the algorithms developed for this problem, several classes of instances are defined. In each class, there is a change in one of the inputs. Processing times are distributed uniformly over two ranges with a mean of 60: (50, 70) and (20, 100). The setup times are uniformly distributed from 12 to 24 which are 20% to 40% of the mean of the processing time. The setup time matrices are asymmetric and satisfy the triangle inequality. The setup time characteristics follow Rios-Mercado and Bard [49].

The problem data can be characterized by three factors, and each of these factors can have at least two levels. These levels are shown in Table 1. Another important issue is the earliest and latest due dates of the jobs. To generate the due dates of jobs, we proposed the following steps:

– Compute total processing time on a machine.

$$P_t = \sum_{i=1}^{n} p_i \tag{8}$$

– Compute total setup time for all possible subsequent jobs and sum it on a machine.

$$S_t = \sum_{i=1}^{n} \sum_{j=1}^{n} s_{ij} \tag{9}$$

Table 1 Factor levels of problem

Factor	Levels		
Number of jobs	6	30	100
Machine distribution	Constant:1	2	10
	Variable: uniform (1, 4)	Uniform (1, 10)	
Processing times	Uniform (50, 70)	Uniform (20, 100)	

– Then d_{j1} and d_{j2} are generated as follows:

- Determine an earliest due date for each job.

$$d_{i1} = \left(\frac{1 + random}{3}\right) \times \left(\frac{S_t}{n} + P_t\right) \Big/ m, \forall i \in N \quad (10)$$

- Determine a latest due date for each job.

$$d_{i2} = d_{i1} + \left(\frac{random}{3} \times \left(\frac{S_t}{n} + P_t\right) \Big/ m\right), \forall i \in N \quad (11)$$

where *random* is a random number from a uniform distribution over range (0.5, 1). This method generates very tight due window interval for each job.

5.2 Hybrid algorithm parameters tuning

It is known that the different levels of the parameters clearly affect the quality of the solutions obtained by a hybrid algorithm. A number of different hybrid algorithms can be obtained with the different combinations of the parameters. We have applied parameters tuning only for the minimum values for trail intensity ($\tau_{\min}$), persisting the trail intensities (ρ) in ACO, initial temperature parameter (φ_1), and final temperature parameter (φ_2) in SA algorithm, considering the following ranges:

- $\tau_{\min}$: two levels (1/7 $\tau_{\max}$ and 1/10 $\tau_{\max}$)
- ρ: three levels (0.7, 0.8 and 0.9)
- φ_1: three levels (0.4, 0.6, and uniform (0.5, 1))
- φ_2: three levels (0.1, 0.01 and uniform (0, 0.1)).

Fifty-four different crosses are obtained by these levels. We generate six instances, two small, two medium, and two large, for each combination of n, m, SDST resulting a total of 252 instances. All 252 instances are solved by 54 different hybrid algorithms.

The results are analyzed by the means of multifactor analysis of variance (ANOVA) technique. (see [44]). Table 2 shows the results for different sizes of problems: small, medium, and large.

Table 2 Parameters tuning

Parameters	Problems		
	Small	Medium	Large
$\tau_{\min}$	1/7 $\tau_{\max}$	1/10 $\tau_{\max}$	1/10 $\tau_{\max}$
ρ	0.9	0.7	0.7
φ_1	Uniform (0.5, 1)	Uniform (0.5, 1)	0.4
φ_2	0.01	0.01	0.01

5.3 Stopping rule

The stopping criterion used when testing all instances with the algorithms is set to a computational time (CPU time) limit fixed to $(m^2 \times (n+1)/2)$ second. This stopping criterion is not only responsive to the number of parallel machines but also is sensitive towards rise in the number of jobs.

5.4 Experimental results

In this section, we are going to compare the proposed hybrid algorithms with the VNS algorithm which proposed by Rocha et al. [50] for the SDST parallel machine scheduling with the objective of minimizing the sum of the completion time plus the sum of the weighted delays. In this paper, VNS algorithm was adapted to the ET problem. The proposed algorithms and VNS algorithm are coded in C++ and run with an Intel Pentium IV dual core 2.5 GHz PC at 896 MB RAM under a Microsoft Windows XP environment.

We use Relative Percentage Index (RPI) as performance measure to compare the methods, because RPI fulfills some drawbacks of relative percentage deviation (RPD) in case of the tardiness objectives. When the ET penalties of each algorithm has been obtained for its instances, the best and worst solutions obtained for each instance (which are named MIN_{sol} and $WORST_{\text{sol}}$, respectively) by any of the four algorithms are calculated. *RPI* is obtained by given formula as follows:

$$RPI = \frac{Alg_{\text{sol}} - Min_{\text{sol}}}{WORST_{sol} - Min_{sol}} \quad (12)$$

where Alg_{sol} is the ET penalties obtained for a given algorithm and instance. *RPI* takes value between 0 and 1. Clearly, lower values of *RPI* are preferred.

5.4.1 Experimental results

The results of the experiments for two subsets, averaged for each one of the n and m configurations, are shown in Table 3. Each instance is solved using six different seeds and the average solution is considered.

As it can be seen, the hybrid algorithm provides better results than other algorithms. In order to verify the statistical validity of the results shown in Table 3 and to confirm which the best algorithm is, we have performed a design of experiments and an ANOVA where we consider the different algorithms as a factor and the response variable RPI.

The results demonstrate that there is a clear statistically significant difference between performances of the algorithms. The means plot and least significant difference

Table 3 Average relative percentage deviation $(\overline{\text{RPI}})$ for algorithms grouped by n and m

Problem size		Algorithm			
Number of job	Number of machine	HMH	SAV	ANV	VNS
6	1	0.09838	0.74112	0.56562	0.61579
	2	0.09245	0.75468	0.45992	0.50747
	Uniform (1, 4)	0.13762	0.71381	0.46910	0.50643
	Uniform (1, 6)	0.11258	0.72304	0.47928	0.53404
	6 Job	0.10785	0.72907	0.50806	0.56129
30	1	0.09143	0.56329	0.76991	0.69430
	2	0.16619	0.55195	0.78134	0.77440
	10	0.04865	0.77529	0.74111	0.74734
	Uniform (1, 4)	0.16224	0.52754	0.77627	0.74659
	Uniform (1, 10)	0.12055	0.59705	0.79841	0.74225
	30 Job	0.09808	0.62332	0.77903	0.72995
100	1	0.06528	0.72822	0.62594	0.59018
	2	0.04952	0.78619	0.67684	0.56156
	10	0.15261	0.68874	0.76454	0.55859
	Uniform (1, 4)	0.05769	0.77425	0.72227	0.59794
	Uniform (1, 10)	0.06064	0.79950	0.68098	0.54634
	100 Job	0.08124	0.75790	0.68455	0.56013
Average		0.09572	0.70343	0.65722	0.61712

(LSD) intervals (at the 95% confidence level) for two algorithms are shown in Fig. 5.

5.4.2 Analysis of controlled factors

Analysis of problem size factor (number of jobs) In order to see the effect of number of jobs on two algorithms, a two-way ANOVA is applied. Means plot and LSD intervals (at the 95% confidence level) for the interaction between the factors of type of algorithm and number of jobs are shown in Fig. 6. As we can see, in all the cases of n=100, n=30, and n=6, the HMH algorithm works better than other algorithms.

Analysis of m factor (number of machines) Another two-way ANOVA and LSD test are applied to see the effect of magnitude of machines on quality of the algorithms. The results are shown in Fig. 7. As we can see, in all the cases, the HMH algorithm works better than others.

6 Conclusions and future works

In this paper, a sequence-dependent setup time parallel machine scheduling problem with earliness and tardiness penalties is attacked by means of a hybrid algorithm approach. The problem involves job specific due window time with sequence-dependent setup time, and this extensively complicates the problem. The objective function includes earliness/tardiness penalties. It is assumed that the ET penalties will not occur if the job is completed within the due window. However, ET penalties will occur if a job is completed outside the due window. We have defined an efficient method to calculate the earliest and latest due date.

This paper suggested hybrid metaheuristic algorithm which combines ACO, SA, and VNS in a population-based context. For each new generation of schedules, the key is to use hybridizing the population-based evolutionary searching ability of ACO with the local improvement ability of some VNS and SA to balance exploration and exploitation.

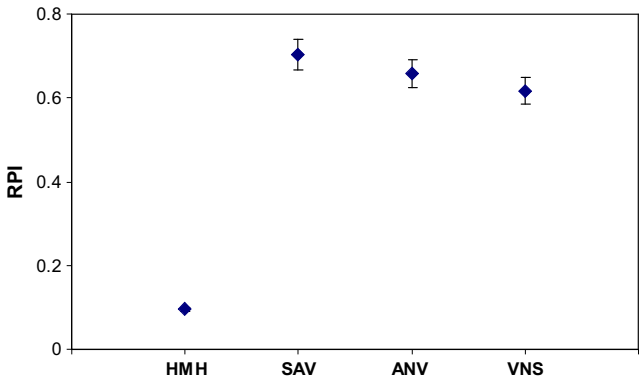

Fig. 5 Plot of $\overline{\text{RPI}}$ for the type of algorithm factor

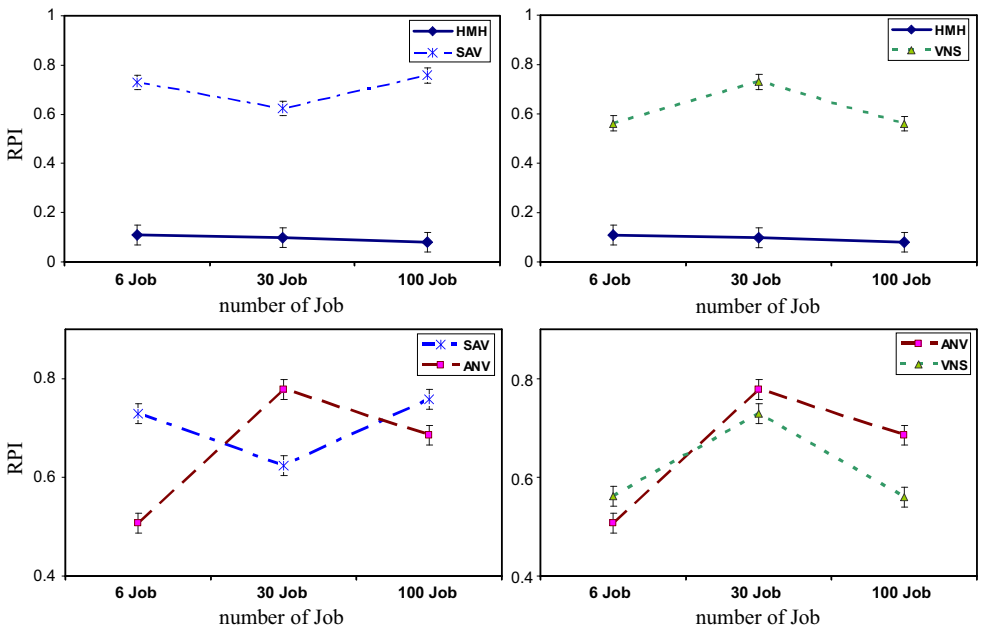

Fig. 6 Plots of $\overline{\text{RPI}}$ for the interaction between the type of algorithm and number of jobs

In the proposed HMH, the balance between the global exploration and the local exploitation was stressed. This approach was generated new individuals that can effectively guide and speed up the search. We believe that the contribution of this method is not only the development of a framework by which due date-related problems can be solved but also the generality of the framework, as it can solve different categories of scheduling problems readily and with very good performance. The flexibility of this algorithm in the exploration is plausible. For example, depending on the number of machines, algorithm can intelligently change local search type defined in VNS algorithm and leading to better results.

Our computational experiments demonstrated that this algorithm yields excellent results, which outperforms the VNS algorithm proposed in a lately published literature for this problem. There are a number of opportunities for future research. Of most significance include generalizations in the model to accommodate (1) multiple objectives, (2) other versions of the scheduling problems, e.g., single machine or job shop scheduling problems, and (3) dynamic job arrivals.

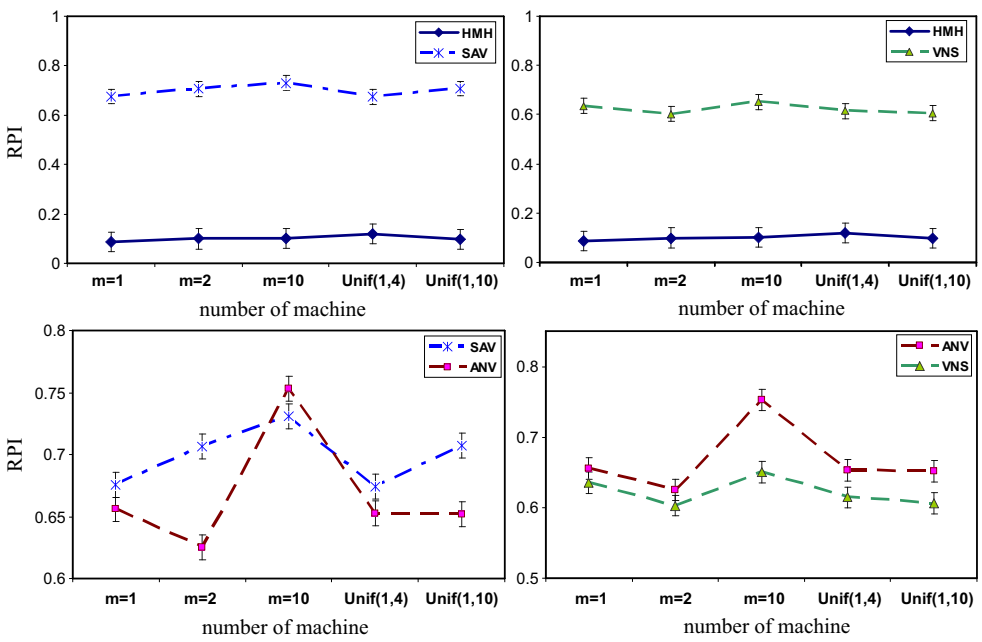

Fig. 7 Plots of $\overline{\text{RPI}}$ for the interaction between the type of algorithm and magnitude of machines

References

1. Aarts E, Lenstra JK (1997) Search in combinatorial optimization. Wiley, New York
2. Ahmed MU, Sundararaghavan PS (1990) Minimizing the weighted sum of late and early completion penalties in a single machine. IIE Trans 22(3):288–290. doi:10.1080/07408179008964183
3. Ahuja RK, Ergun O, Orlin JB, Punnen AP (2002) A survey of very large-scale neighborhood search techniques. Discrete Appl Math 123:75–102. doi:10.1016/S0166-218X(01)00338-9
4. Almeida MT, Centeno M (1998) A composite heuristic for the single machine early/tardy job scheduling problem. Comput Oper Res 25:625–635. doi:10.1016/S0305-0548(97)00097-X
5. Anger FD, Lee CY, Martin-Vega LA (1986) Single-machine scheduling with tight windows. Research Paper, 86–16, University of Florida
6. Anghinolfi D, Paolucci M (2007) Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. Comput Oper Res 34:3471–3490. doi:10.1016/j.cor.2006.02.009
7. Balakrishnan N, Kanet JJ, Sridharan SV (1999) Early/tardy scheduling with sequence-dependent setups on uniform parallel machines. Comput Oper Res 26:127–141. doi:10.1016/S0305-0548(98)00051-3
8. Bean JC (1994) Genetic algorithms and random keys for sequencing and optimization. ORSA J Comput 6(2):154–160
9. Bilge U, Kirac F, Kurtulan M, Pekgun PA (2004) Tabu search algorithm for parallel machine total tardiness problem. Comput Oper Res 31:397–414. doi:10.1016/S0305-0548(02)00198-3
10. Birman M, Mosheiov G (2004) A note on a due-date assignment on a two-machine flow-shop. Comput Oper Res 31:473–480. doi:10.1016/S0305-0548(02)00225-3
11. Bülbül K, Kaminsky P, Yano C (2007) Preemption in single machine earliness/tardiness scheduling. J Sched 10:271–292. doi:10.1007/s10951-007-0028-6
12. Chang P-C, Chen S-H, Fan C-Y (2008) A hybrid electromagnetism-like algorithm for single machine scheduling problem. Expert Syst Appl 36:1259–1267. doi:10.1016/j.eswa.2007.11.050
13. Chen ZL (1996) Scheduling and common due date assignment with earliness–tardiness penalties and batch delivery costs. Eur J Oper Res 93:49–60. doi:10.1016/0377-2217(95)00133-6
14. Chen ZL, Lee CY (2002) Parallel machine scheduling with a common due window. Eur J Oper Res 136:512–527. doi:10.1016/S0377-2217(01)00068-6
15. Chen ZL, Powell WB (1999) A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. Eur J Oper Res 116:221–233
16. Cheng TCE, Chen ZL (1994) Parallel-machine scheduling with earliness and tardiness penalties. J Oper Res Soc 45:685–695
17. Dorigo M, Stuetzle T (2004) Ant colony optimization. MIT, Boston, MA
18. Emmons H (1987) Scheduling to a common due-date on parallel uniform processors. Nav Res Logistics Q 34:803–810. doi:10.1002/1520-6750(198712)34:6<803::AID-NAV3220340605>3.0.CO;2-2
19. Esteve B, Aubijoux C, Chartier A, Tkindt V (2006) A recovering beam search algorithm for the single machine just-in-time scheduling problem. Eur J Oper Res 172:798–813. doi:10.1016/j.ejor.2004.11.014
20. Federgruen A, Mosheiov G (1997) Heuristics for multi-machine min–max scheduling problems with general earliness and tardiness costs. Nav Res Logistics 44:287–299. doi:10.1002/(SICI)1520-6750(199704)44:3<287::AID-NAV4>3.0.CO;2-4
21. Feng G, Lau HC (2005) Efficient algorithms for machine scheduling problems with earliness and tardiness penalties. In: Proceedings of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications, New York, USA, July 18–21, 2005, pp 196–211
22. Fowler JW, Horng SM, Cochran JK (2003) A hybridized genetic algorithm to solve parallel machine scheduling problems with sequence-dependent setups. Int J Ind Eng Theory Appl Pract 10:232–243
23. Gajpal Y, Rajendran C (2006) An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops. Int J Prod Econ 101:259–272. doi:10.1016/j.ijpe.2005.01.003
24. Gendreau M, Laporte G, Guimaraes EM (2001) A divide and merge heuristic for the multiprocessor scheduling problem with sequence-dependent setup times. Eur J Oper Res 133:183–189. doi:10.1016/S0377-2217(00)00197-1
25. Gordon V, Proth JM, Chu C (2002) A survey of the state-of-the-art of common due-date assignment and scheduling research. Eur J Oper Res 135:1–25. doi:10.1016/S0377-2217(01)00181-3
26. Hansen P, Mladenovic N, Dragan U (2004) Variable neighborhood search for the maximum clique. Discrete Appl Math 145 (1):117–125. doi:10.1016/j.dam.2003.09.012
27. Heady RB, Zhu Z (1998) Minimizing the sum of job earliness and tardiness in a multimachine system. Int J Prod Res 36:1619–1632. doi:10.1080/002075498193192
28. Herrmann JW, Lee CY (1993) On scheduling to minimize earliness–tardiness and batch delivery costs with a common due date. Eur J Oper Res 70:272–288. doi:10.1016/0377-2217(93)90239-J
29. Hiraishi K, Levner E, Vlach M (2002) Scheduling of parallel identical machines to maximize the weighted number of just-in-time jobs. Comput Oper Res 29:841–848. doi:10.1016/S0305-0548(00)00086-1
30. Hoogeveen JA (2005) Multicriteria scheduling. Eur J Oper Res 167:592–623. doi:10.1016/j.ejor.2004.07.011
31. Janiak A, Kozan E, Lichtenstein M, Oguz C (2007) Metaheuristic approaches to the hybrid flowshop scheduling problem with a cost-related criterion. Int J Prod Econ 105:407–424. doi:10.1016/j.ijpe.2004.05.027
32. Kim D, Kim K, Jang W, Chen F (2002) Unrelated parallel machine scheduling with setup times using simulated annealing. Comput Integr Manuf 18:223–231. doi:10.1016/S0736-5845(02)00013-3
33. Kima DW, Na DG, Chenb FF (2003) Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. Comput Integr Manuf 19:173–181. doi:10.1016/S0736-5845(02)00077-7
34. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680. doi:10.1126/science.220.4598.671
35. Kramer FJ, Lee CY (1994) Due window scheduling for parallel machines. Math Comput Model 20:69–89. doi:10.1016/0895-7177(94)90208-9
36. Kubiak W, Lou S, Sethi R (1990) Equivalence of mean flow time problems and mean absolute deviation problems. Oper Res Lett 9:371–374. doi:10.1016/0167-6377(90)90056-B
37. Kurz ME, Askin RG (2001) Heuristic scheduling of parallel machines with sequence-dependent setup times. Int J Prod Res 39:3747–3769. doi:10.1080/00207540110064938
38. Lam K, Xing W (1997) New trends in parallel machine scheduling. Int J Oper Manage 17:326–338. doi:10.1108/01443579710159932
39. Lauff V, Werner F (2004) Scheduling with common due date, earliness and tardiness penalties for multimachine problems: a survey. Math Comput Model 40:637–655. doi:10.1016/j.mcm.2003.05.019
40. Lee CY, Danusaputro S, Lin CS (1991) Minimizing weighted number of tardy jobs and weighted earliness–tardiness penalties about a common due date. Comput Oper Res 18:379–389. doi:10.1016/0305-0548(91)90098-C

41. Lenstra J, Rinnooy Kan A, Brucker P (1977) Complexity of machine scheduling problems. Ann Discrete Math 1:343–362. doi:10.1016/S0167-5060(08)70743-X
42. Logendrana R, Mcdonellb B, Smuckera B (2007) Scheduling unrelated parallel machines with sequence-dependent setups. Comput Oper Res 11:3420–3438. doi:10.1016/j.cor.2006.02.006
43. Mladenovic N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24:1097–1100. doi:10.1016/S0305-0548(97)00031-2
44. Montgomery DC (2000) Design and analysis of experiments, 5th edn. Wiley, New York
45. Nessah F, Yalaoui F, Chu C (2005) New heuristics for identical parallel machine scheduling with sequence-dependent setup times and dates. In: Proceedings of the International Conference on Industrial Engineering and Systems Management, Marrakech, Morocco, May 16–19, 2005, pp 32–41
46. Park Y, Kim S, Lee YH (2000) Scheduling jobs on parallel machines applying neural network and heuristic rules. Comput Ind Eng 38:189–202. doi:10.1016/S0360-8352(00)00038-3
47. Pinedo M (1995) Scheduling theory, algorithms, and systems. Prentice-Hall, Englewood Cliffs, NJ
48. Radhakrishnan S, Ventura JA (2000) Simulated annealing for parallel machine scheduling with earliness–tardiness penalties and sequence-dependent setup times. Int J Prod Res 38:2233–2252. doi:10.1080/00207540050028070
49. Rios-Mercado RZ, Bard JF (1998) Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. Comput Oper Res 25(5):351–366. doi:10.1016/S0305-0548(97)00079-8
50. Rocha M, Gómez Ravetti M, Mateus GR, Pardalos PM (2007) Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighborhood search. IMA J Manage Math 18:101–115. doi:10.1093/imaman/dpm016
51. Sivrikaya-Serifoglu F, Ulusoy G (1999) Parallel machine scheduling with earliness and tardiness penalties. Comput Oper Res 26:773–787. doi:10.1016/S0305-0548(98)00090-2
52. Stützle T (1998) An ant approach for the flowshop problem. In: Zimmerman H (ed) Proceedings of the Sixth European Congress on Intelligent Techniques and Soft Computing (EUFIT'98), vol 3. Mainz, Aachen, Germany, pp 1560–1564
53. Tahar DN, Yalaoui F, Chu C, Amodeo L (2006) A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times. Int J Prod Econ 99:63–73. doi:10.1016/j.ijpe.2004.12.007
54. Talbi E (2002) A taxonomy of hybrid metaheuristics. J Heuristics 8(5):541–564. doi:10.1023/A:1016540724870
55. Tamimi SA, Rajan VN (1997) Reduction of total weighted tardiness on uniform machines with sequence-dependent setups. In: Proceedings of the 6th Industrial Engineering Research Conference, pp 181–185
56. Tasgetiren MF, Sevkli M, Liang YC, Gencyilmaz G (2007) Particle swarm optimization algorithm for makespan and total flowtime minimization in permutation flowshop sequencing problem. Eur J Oper Res 177(3):1930–1947. doi:10.1016/j.ejor.2005.12.024
57. Tian P, Ma J, Zhang DM (1999) Application of the simulated annealing algorithm to the combinatorial optimization problem with permutation property: an investigation of generation mechanism. Eur J Oper Res 118:81–94. doi:10.1016/S0377-2217(98)00308-7
58. Vignier A, Sonntag B, Portmann MC (1999) Hybrid method for a parallel-machine scheduling problem. In: IEEE Symposium on Emerging Technologies and Factory Automation, ETFA, 1, 671–678
59. Wan G, Yen BPC (2002) Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. Eur J Oper Res 142:271–281. doi:10.1016/S0377-2217(01)00302-2
60. Yao X (1995) A new simulated annealing algorithm. Int J Comput Math 56:161–168. doi:10.1080/00207169508804397
61. Yeung WK, Oğuz C, Cheng TCE (2004) Two-stage flowshop earliness and tardiness machine scheduling involving a common due window. Int J Prod Econ 90:421–434. doi:10.1016/S0925-5273(03)00044-6