Int. J. Appl. Math. Comput. Sci., 2020, Vol. 30, No. 3, 573–584
DOI: 10.34768/amcs-2020-0042

TWO META–HEURISTIC ALGORITHMS FOR SCHEDULING ON UNRELATED MACHINES WITH THE LATE WORK CRITERION

WEN WANG [a,b], XIN CHEN [a,*], JEDRZEJ MUSIAL [b], JACEK BLAZEWICZ [b,c]

[a]School of Electronics and Information Engineering
Liaoning University of Technology
Shiying 169, 121001 Jinzhou, China
e-mail: chenxin.lut@hotmail.com

[b]Institute of Computing Science
Poznań University of Technology
ul. Piotrowo 3a, 60-965 Poznań, Poland

[c]Institute of Bioorganic Chemistry
Polish Academy of Sciences
ul. Noskowskiego 12, 61-704 Poznań, Poland

A scheduling problem in considered on unrelated machines with the goal of total late work minimization, in which the late work of a job means the late units executed after its due date. Due to the NP-hardness of the problem, we propose two meta-heuristic algorithms to solve it, namely, a tabu search (TS) and a genetic algorithm (GA), both of which are equipped with the techniques of initialization, iteration, as well as termination. The performances of the designed algorithms are verified through computational experiments, where we show that the GA can produce better solutions but with a higher time consumption. Moreover, we also analyze the influence of problem parameters on the performances of these meta-heuristics.

Keywords: late work minimization, unrelated machines, tabu search, genetic algorithm.

1. Introduction

Late work minimization is one of the classical problems in the scheduling field. It was first introduced in 1984 (Błażewicz, 1984), and has attracted more attention in the recent years (e.g., Chen *et al*., 2017; Gerstl *et al*., 2019; Piroozfard *et al*., 2018). Basically, a job's late work stands for the penalty when it is completed after an expected time, i.e., the value equals the late units executed after its due date. Since this parameter can model any situation where a perishable commodity is involved (Potts and Van Wassenhove, 1992b), scheduling with late work minimization was widely studied in the past four decades (Abasian *et al*., 2014; Błażewicz and Finke, 1987; Kovalyov *et al*., 1994; Sterna, 2007).

Meta-heuristic algorithms (Blazewicz *et al*., 2008; Talbi, 2009) are a kind of commonly used approach to solve the intractable (NP-hard) problems. Instead of searching for an optimal solution with a high time consumption (even for small problem instances), they can always produce a good feasible solution quite quickly. Among others, tabu search (TS) (Glover, 1989; 1990; Rybarczyk *et al.*, 2017) and genetic algorithms (GAs) (Holland, 1962; Whitley, 1994) are two classical ones, which are generally used for solving the optimization problems such as scheduling (e.g., Blazewicz *et al*., 2000; Kundakcı and Kulak, 2016; Servranckx and Vanhoucke, 2019), Internet shopping (e.g., Błażewicz *et al*., 2010; Lopez-Loces *et al*., 2016), or routing (e.g., Labib *et al*., 2019; Yan *et al*., 2013).

In this paper, we study the late work scheduling problem on unrelated parallel machines, which is NP-hard since its special case (scheduling on identical machines with a common due date) is already NP-hard in the strong sense (Chen *et al*., 2016). To solve the problem efficiently,

*Corresponding author

we design two meta-heuristic algorithms for it, namely, a TS and a GA. Finally, the proposed methods are analyzed by computational experiments, in which we compare the two algorithms from the viewpoints of solution quality, as well as execution time.

The rest of this paper is organized as follows. In Section 2 we give the definition of the problem considered, and discuss the related work. Sections 3 and 4 are devoted to the design of the two meta-heuristics, i.e., TS and the GA, respectively, which are evaluated by the computational experiments in Section 5. Finally, we give the conclusions and point out some future work in Section 6.

2. Problem definition and related work

2.1. Problem definition. As we have mentioned before, the late work of a job J_j (denoted as Y_j) equals the late part executed after its due date (d_j), i.e.,

$$Y_j = \begin{cases} 0, & C_j \le d_j, \\ C_j - d_j, & d_j < C_j \le d_j + p_j, \\ p_j, & C_j \ge d_j + p_j, \end{cases}$$

where p_j and C_j are the processing and completion time of J_j, respectively. Thus, the problem we consider in this paper can be defined as follows.

Input: Given m parallel unrelated machines $M = \{M_1, M_2, \ldots, M_m\}$ and n jobs $J = \{J_1, J_2, \ldots, J_n\}$. The processing time of job J_j ($1 \le j \le n$) on machine M_i ($1 \le i \le m$) is $p_{j,i}$, and each job has an arbitrary due date d_j. Consequently, if job J_j was scheduled on machine M_i, its late work could be calculated as $Y_j = \min\{\max\{0,\ C_j - d_j\}, p_{j,i}\}$.

Output: Schedule J on M without preemption to minimize the total late work of all jobs, i.e., $Y = \sum_{j=1}^{n} Y_j$.

Using the three-field notation of scheduling models (Graham *et al.*, 1979), the problem can be denoted as $R||Y$.

2.2. Related work. The parameter of late work was first proposed by Błażewicz (1984), who was motivated by the information collection in a control system. Therefore, he used the term of "information loss" at that time. Later, Potts and Van Wassenhove (1992b) analyzed the characteristics of this concept again, and suggested to use a more general phrase, i.e., "late work." After these pioneers, scheduling with the late work criterion has been widely studied among single, shop, as well as parallel environments.

For scheduling on a single machine, Potts and Van Wassenhove (1992b) showed that problem $1||Y$ is NP-hard, and designed a dynamic programming algorithm which runs in pseudo-polynomial time to solve it optimally. Almost at the same time, the two authors (Potts and Van Wassenhove, 1992a) also proposed two fully polynomial-time approximation schemes (FPTASs) to solve this problem ($1||Y$) approximatively. Later, their results were extended to weighted model, i.e., problem $1||Y_w$ (Hariri *et al.*, 1995; Kovalyov *et al.*, 1994). Lin and Hsu (2005) studied the problems with release time, in which they proposed a branch-and-bound (B&B) algorithm for the general case ($1|r_j|Y$), and several polynomial-time algorithms for some special cases ($1|r_j,\ pmtn|Y$ or $1|r_j, d_j = d|Y$, respectively). Wu *et al.* (2016) studied a single-machine scheduling problem with a position-based learning effect ($1|LE|Y$), and developed a B&B algorithm and three heuristic-based genetic algorithms to solve this problem. Recently, Chen *et al.* (2019) introduced the deadline restriction into the original model, i.e., a series of $1|\overline{d}|Y_w$ related problems, to study their complexities, approximabilities, as well as polynomially solvablities for some special cases.

The investigations on late work minimization in shop systems started by Błażewicz *et al.* after 2000, when they showed that problems $O2|d_j = d|Y_w$ (Błażewicz *et al.*, 2004), $F2|d_j = d|Y_w$ (Błażewicz *et al.*, 2005) and $J2|d_j = d,\ n_j \le 2|Y_w$ (Błażewicz *et al.*, 2007) are all NP-hard in the weak sense. Following their work, Lin *et al.* (2006) showed the unweighted problem $F2|d_j = d|Y$ is also NP-hard, and proposed a B&B algorithm for its general case $F2||Y$. Later, Pesch and Sterna (2009) studied a more general case $F|r_j|Y$, where they used a genetic algorithm to solve the problem. Then, Chen *et al.* (2017) revisited the complexity of flow shop scheduling with the late work criterion, in which they showed that the problem is NP-hard in the strong sense if the number of machines is greater than or equal to 3, i.e., $Fm|d_j = d|Y$ for $m \ge 3$. Moreover, they introduced a learning effect into this model, and designed a particle swarm optimization algorithm for it ($F|LE|Y$). Recently, Gerstl *et al.* (2019) studied two versions of the m-machine proportionate flow shop scheduling problem with late work minimization ($F|p_{ij} = p_j|Y$), and introduced two pseudo-polynomial dynamic programming methods for them, respectively.

The late work parameter was first introduced in the context of parallel machines scheduling. In its seminal paper, Błażewicz (1984) showed that problems $P|r_j,\ pmtn|Y_w$ and $Q|r_j,\ pmtn|Y_w$ are polynomially solvable, while the non-preemption version $P||Y$ is NP-hard. Then, Leung (2004) designed an $O(n^2 log^3 n)$ and an $O(m^2 n^4 logmn)$ algorithm for $P|r_j,\ pmtn|Y_w$ and $Q|r_j,\ pmtn|Y_w$, respectively. He also showed that if the weights of jobs are not considered, the complexities could be reduced to $O(n^2 log^2 n)$ and $O(m^2 n^3 logmn)$, respectively. Abasian *et al.* (2014) introduced the

communication delays into the model, to study the problem $Pm|prec,\ comu|Y_w$. The authors constructed an integer linear mathematical programming model, as well as proposed a B&B algorithm for it. Recently, Chen *et al.* (2020b) proposed an FPTAS for parallel machines scheduling problem, with the context of early work maximization. Then the relationships between the two relevant criteria, i.e., early and late work were further analyzed (Chen *et al.*, 2020a). To understand this topic more exactly, we propose to refer to the survey paper by Sterna (2011) or the latest monograph by Błażewicz *et al.* (2019).

3. Tabu search for $R||Y$

In this section, we propose a tabu search (TS) algorithm for problem $R||Y$. A tabu search (for any combinatorial optimization problem) always begins with a pre-defined or randomly selected feasible solution, and then explores its neighborhood to generate a new solution. During the explorations, a tabu list is used to avoid cycles, i.e., repeated searching (but sometimes a tabu action may be re-called if some aspiration criteria are met). Finally, tabu search stops when the termination criteria are satisfied, and outputs the best-found solution during the iterations.

In our TS, we use a vector with $n+m-1$ elements to represent a feasible solution, composed of n positive integers from 1 to n to indicate jobs' indices, and $m-1$ negative integers from $-(m-1)$ to -1 to divide these jobs into m machines. For example, when $n=7$ and $m=3$, the schedule corresponding to the encoding $(3,5,4,-1,7,1,-2,6,2)$ is

$$M_1 : J_3 \prec J_5 \prec J_4,$$
$$M_2 : J_7 \prec J_1,$$
$$M_3 : J_6 \prec J_2,$$

in which we use $J_j \prec J_k$ to express that J_j is scheduled preceding J_k on the same machine.

3.1. Initial solution. A reasonable initial solution has great influence on the final result of TS. In this paper, we use 9 heuristic policies plus a random way to generate 10 feasible solutions, among of which the best one (with the minimum criterion value) will be selected to represent the initial solution. All of the 9 heuristics contain two parts and can be denoted as A-B, in which part A means to sort jobs into a list by a pre-defined order, while part B assigns jobs in this list one by one, based on some machine selection rule. The details of part A and B are enumerated in Tables 1 and 2, respectively. In summary, the nine heuristic policies are EDD-$MinC$, EDD-$MinY$, EDD-$MinP$, SPT-$MinC$, SPT-$MinY$, SPT-$MinP$, LPT-$MinC$, LPT-$MinY$ and LPT-$MinP$.

Table 1. Rules in Part A.

Rule	Explanation
EDD	Earliest due date first (non-decreasing order of d_j)
SPT	Smallest processing time first (non-decreasing order of $\sum_{i=1}^{m} p_{j,i}$)
LPT	Largest processing time first (non-increasing order of $\sum_{i=1}^{m} p_{j,i}$)

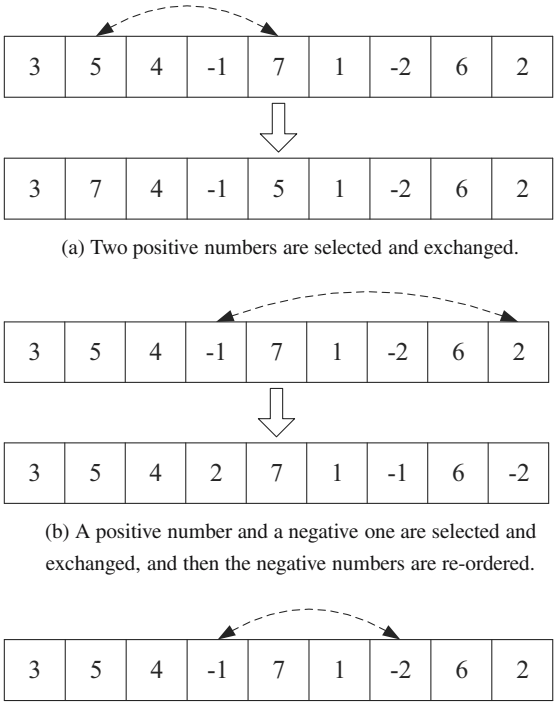
Table 2. Rules in Part B.

Rule	Explanation
$MinC$	Assign the current job to a suitable machine to keep the minimum makespan.
$MinY$	Assign the current job to a suitable machine to keep the minimum current total late work.
$MinP$	Assign the current job to the machine with the minimum processing time, i.e., $\min_{1\le i\le m}\{p_{j,i}\}$.

Normally, we should run the ten ways above (9 heuristics and one random) and choose the best one as the initial solution. However, based on our initial experiments, we found that SPT-$MinP$ beats the other nine in most cases (see $Section$ 5.2 for details). Therefore, we decide to use SPT-$MinP$ directly to generate an initial solution for our TS, to save its running time.

3.2. Iteration. The current solution could generate its offspring by exchanging a pair of elements in its encoding. That is, we define a generation process in TS as follows: (i) two positions are randomly selected based on the current encoding; (ii) if both of the elements on these two positions are negative integers, we omit this selection and do it again (since this schedule is the same as the current one); (iii) otherwise, we exchange these two elements, and re-set the negative integers from -1 to $-(m-1)$ on their original positions if they violate the decreasing order. Then we put this offspring into the neighborhood as one of the candidates for the next iteration. The above processes are illustrated in Fig. 1.

Observe that there are in total $C^2_{n+m-1} = (n+m-1)(n+m-2)/2$ choices if we select two elements from the $(n+m-1)$ vector. However, it is time-consuming and not necessary to try all the possibilities. Therefore, we set the size of neighborhood to be 40 (after a parameter tuning process in Section 5.3.1), i.e., we generate 40 offspring as the candidates and choose the best one (with the minimal criterion value) into the next iteration.



(a) Two positive numbers are selected and exchanged.

(b) A positive number and a negative one are selected and exchanged, and then the negative numbers are re-ordered.

(c) Two negative numbers are selected, so this selection has to be omitted.

Fig. 1. Illustrations of the generation processes.

3.3. Tabu list. A particular characteristic of tabu search is the use of memory, i.e., it uses tabu list(s) to store the information related to the searching process. In our TS, we define a short-term memory and a long-term memory to remember the recently visited solutions. The short-term memory stores all the generated solutions if it still has space (including all the offspring during the iterations), while long-term one stores only the "current solutions" which have been selected for generations. Of course, storing all the visited solutions is time and space consuming (and it makes no sense), so we define the sizes of both memories. Then, both of them adopt the "First In, First Out (FIFO)" strategy, that is, if there is no space for the new generated solution(s), the first one(s) will be removed out and the new one(s) will be put in.

3.4. Termination criteria. Our TS stops if one of the following criteria is satisfied:

(i) the number of iterations achieves a pre-defined quantity (M_ITER);

(ii) the best-found solution has not been changed for a pre-defined period (M_UNCH).

Then, the best-found solution will be output as the final solution of TS.

Algorithm 1. TS.

```
1: Run SPT-MinP to get an initial schedule, denoted as current_solution
2: Put current_solution into long-term memory
3: best_found_solution = current_solution
4: iter = unch = 0
5: while (iter ≤ M_ITER && unch ≤ M_UNCH) do
6:    iter++
7:    Generate 40 un-taboo offspring based on current_solution
8:    Put them into short-term memory
9:    Let the best one among them be best_offspring
10:   if (best_offspring beats best_found_solution) then
11:      unch = 0
12:      best_found_solution = best_offspring
13:   else
14:      unch++
15:   end if
16:   current_solution = best_offspring
17: end while
18: Output best_found_solution
```

3.5. Parameter setting and framework of TS. Based on several preliminary experiments (see details in Section 5.3.1), we set all the parameters in our TS in Table 3. Then, the framework of this meta-heuristics is given as Algorithm 1.

4. Genetic algorithm for $R||Y$

Tabu search is a typical single-solution based meta-heuristics, that is, the generation in each iteration is based on a single solution. In contrast, the genetic algorithm is a classical population-based method, in which a population is used to generate offspring during the iterations.

Inspired by the adaptive process of the natural world, a genetic algorithm can gradually approach the optimal solution(s) (for any optimization problem) by an evolution procedure. Based on an initial population, which contains several chromosomes and each chromosome represents a feasible solution, the algorithm starts the evolution by the operations of selection, crossover and mutation. After several iterations, the algorithm stops when the termination criteria are met.

In this section, we design a genetic algorithm (GA) for problem $R||Y$. The chromosome is encoded in the same way as in TS, i.e., we use an $(n + m - 1)$ vector to represent a feasible schedule. Moreover, we define the operations specifically for the problem, such as crossover, mutation, and so on. Finally, we give the termination criteria of the GA to stop its iterations and output the best-found solution.

Algorithm 2. Select a GA subprocedure.

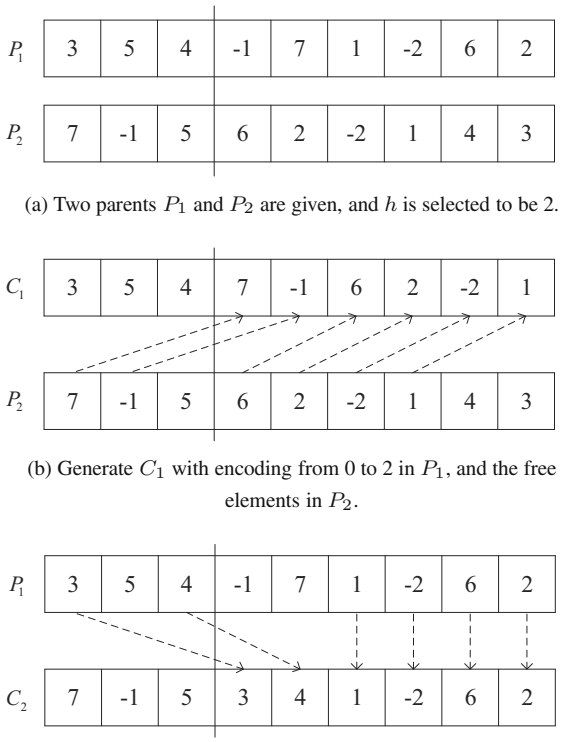
Require: Current population (P) with 200 individuals

Ensure: A selected group (G) with 200 individuals for crossover and mutation

1: Calculate the fitness of each individual I_k ($1 \leq k \leq 200$) in P, denoted as $f(I_k)$
2: Put the best 40 individuals with higher fitness values into G directly {best-reserved}
3: Construct a roulette wheel. For individual I_k, its possibility is set to be $f(I_k)/\sum_{k=1}^{200} f(I_k)$
4: Run the roulette wheel 160 times. Each time, pick up an individual and copy it into G {roulette wheel}

Table 3. Parameter setting in TS.

Parameter	Value
Size of neighborhood	40
Size of short-term memory	100
Size of long-term memory	50
M_ITER	1000
M_UNCH	100



(a) Two parents P_1 and P_2 are given, and h is selected to be 2.

(b) Generate C_1 with encoding from 0 to 2 in P_1, and the free elements in P_2.

(c) Generate C_2 with encoding from 0 to 2 in P_2, and the free elements in P_1.

Fig. 2. Illustrations of the crossover in the GA.

4.1. Initial population. We define the population size to be 200 (after parameter tuning in Section 5.3.2), that is, there are 200 chromosomes in our GA. Among them, 9 chromosomes are initialized by the heuristic policies in Section 3.1, which are $EDD\text{-}MinC$, $EDD\text{-}MinY$, $EDD\text{-}MinP$, $SPT\text{-}MinC$, $SPT\text{-}MinY$, $SPT\text{-}MinP$, $LPT\text{-}MinC$, $LPT\text{-}MinY$ and $LPT\text{-}MinP$, respectively; while the other 191 are initialized at random.

It is worth mentioning that, although we know that $SPT\text{-}MinP$ works better than the other heuristics (see details in $Section$ 5.2), we still generate them as well extend the searching range of our GA.

4.2. Operations during iteration.

4.2.1. Selection. We first define a fitness function for the individuals (chromosomes) in the population to evaluate their qualities, so that we can make a relevant choice during selection. Since the problem considered is a minimization problem, we use the difference between a large number and its criterion value as an individual's fitness. That is, for an individual I_k ($1 \leq k \leq 200$), the fitness function is

$$f(I_k) = M - Y(I_k),$$

where M is a sufficiently large number so that every individual has a positive fitness, and $Y(I_k)$ is the total late work of the schedule corresponding to I_k.

According to the phenomenon of survival of the fittest, a better individual in the current generation has a higher possibility to generate its offspring (the roulette wheel selection). Moreover, to guarantee a good evolution of the whole population, we select the best 20% individuals without roulette wheel. That is, our selection process is based on a roulette wheel mixed with a best-reserved policy (see Algorithm 2 for details).

Note that there could be repeated individual(s) in G, but we agree this overlap since it is a "natural selection" process.

4.2.2. Crossover. The crossover operation is defined to form an evolutionary process towards optimal solutions, in which the offspring inherit superior genes from their parents. We adopt a single-point crossover technique in our GA, which is described as follows: (i) given two parents P_1 and P_2, randomly choose a positive integer $h \in [0,\ n+m-2]$; (ii) keep the encoding on positions from 0 to h in P_i ($i \in \{1,\ 2\}$) as they are, and set the positions from $h+1$ to $n+m-2$ in P_i with the free elements in P_{3-i} one by one, where a free element means it does not appear in P_i from position 0 to h for the moment; (iii) output two new individuals as the offspring of P_1 and P_2, denoted as C_1 and C_2, respectively. An illustration of the crossover operation is shown in Fig. 2.

To keep the stabilization, we introduce a crossover probability p_c in the GA. That is, if the probability is hit,

Algorithm 3. Cross a GA subprocedure.

Require: A group G with 200 individuals after selection
Ensure: A medial group $G^{'}$ with 200 individuals for mutation
1: Partition G into 100 pairs randomly
2: For each pairs, do the crossover with a probability p_c, and put their offspring into $G^{'}$

Table 4. Parameter setting in the GA.

Parameter	Value
Population size	200
Crossover probability p_c	0.95
Mutation probability p_m	0.2
M_ITER	300
M_UNCH	50

we output C_i based on the above process; otherwise, we just output C_i exactly the same as P_i (for $i \in \{1, 2\}$). Therefore, the whole process of crossover is shown in Algorithm 3.

4.2.3. Mutation. The mutation operation in the genetic algorithm is for the randomness during the exploration. In our GA, we use the same method as neighborhood searching in TS (in Section 3.2) to execute this operation, with a probability denoted as p_m.

Therefore, the process of mutation could be described as follows: for each individual in $G^{'}$ (output by crossover), do an exchange (only once) for a random pair of the elements in its encoding, with a probability p_m. Then output the changed group, denoted as $G^{''}$.

4.2.4. Elitist preservation. Finally, to keep the convergence of the iteration procedure, we add an elitist strategy in our GA. That is, if the best individual (denoted as *current_best*) in the current population P beats the best one in group $G^{''}$ (with a smaller criterion value, or equally, a bigger fitness value), we remove the worst one from $G^{''}$, and put *current_best* into $G^{''}$. Then, a new population is formed for the next generation.

4.3. Termination criteria. We use the same idea in TS to define the GA's termination conditions, as follows:

(i) the process of generation reaches a pre-defined number (M_ITER);

(ii) the elitist preservation strategy has been applied for several times (M_UNCH) continuously (which means that the best-found solution has not been improved for a period).

However, in contrast to TS, we do not have to remember the best-found solution during the whole exploration. Due to the elitist preservation strategy, the best individual in the final generation is the best one during the searching process, and should be output as the final result of the GA.

Algorithm 4. GA.

1: Initialize a population, denoted as P
2: *iter* = *unch* = 0
3: **while** (*iter* $\leq$ M_ITER && *unch* $\leq$ M_UNCH) **do**
4: *iter*++
5: Find the best individual in P, denoted as *current_best*
6: G = select(P)
7: $G^{'}$ = cross(G)
8: $G^{''}$ = mutate($G^{'}$)
9: **if** (*current_best* beats the best one in $G^{''}$) **then**
10: *unch*++
11: P = elitist_preserve(*current_best*, $G^{''}$)
12: **else**
13: *unch* = 0
14: $P = G^{''}$
15: **end if**
16: **end while**
17: Output *current_best* in P

4.4. Parameter setting and the framework of the GA. The parameters in the GA are set in Table 4 after an initial experiment (in Section 5.3.2), and the framework of this algorithm is then given below (Algorithm 4).

It is worth mentioning that, in order to speed up the running process of the GA (for comparing it with TS), we use a single-point crossover operation in it, which may lead to a convergence into local optimum with a high probability. To avoid this phenomenon and extend the exploration, we set the mutation probability (p_m) as a rather high value, i.e., 0.2 in our GA.

5. Computational experiments

5.1. Test data. We generate test data according to the frame proposed by Lin *et al.* (2006), which is a widely used benchmark in the late work scheduling field. However, since this method was originally designed for a two-machine flow shop system ($F2||Y$), we have to change it somehow to fit the problem considered in this paper ($R||Y$).

The processing time $p_{j,i}$ was generated with a discrete uniform distribution from [1, 10], and a regular processing time of job J_j, denoted as $\overline{p}_j$, was set as

$$\overline{p}_j = \frac{1}{m}\sum_{i=1}^{m} p_{j,i}.$$

Next, we sort these jobs by the non-decreasing order of

Table 5. Parameter values in TS.

Parameter	Small value	Big value
Size of neighborhood	20	40
Size of short-term memory	50	100
Size of long-term memory	50	100

$\overline{p}_j$. For convenience, we still use J_j to denote the j-th job after sorting (and $\overline{p}_j$ for its regular processing time as well). Then, the due date of J_j, i.e., d_j, was generated randomly from the interval

$$(\overline{p}_j, \overline{p}_j + \frac{1}{m\beta}\sum_{k=1}^{j}\overline{p}_{n-k+1}],$$

in which β is a tightness controller and set to be 3, 5 and 7, respectively. Moreover, the jobs and machines numbers were set as $n \in \{100,\ 300,\ 500\}$ and $m \in \{3,\ 5,\ 7,\ 10\}$, respectively.

All the algorithms were implemented in C++ with the IDE of Visual Studio 2017, and run on the platform with Intel Core i7 8565U 1.80 GHz CPU and 8 GB RAM.

5.2. Initial experiments for the simple heuristics. As we have mentioned in Section 3.1, there could be 9 simple heuristics plus one random way to generate feasible schedules, which could be used as the initial solutions of TS. One of an alternative approach is to run these methods respectively, and choose the best one to be the initial solution. However, if we knew in advance that one of the policies works much better than the others, we could use it directly to avoid time consumptions on the other procedures.

With this motivation, we first do an initial experiment to compare the performances of these simple heuristics, which are $EDD\text{-}MinC$, $EDD\text{-}MinY$, $EDD\text{-}MinP$, $SPT\text{-}MinC$, $SPT\text{-}MinY$, $SPT\text{-}MinP$, $LPT\text{-}MinC$, $LPT\text{-}MinY$, $LPT\text{-}MinP$, as well as the random way. For each combination of (m, n, β), we randomly generated 20 instances, and the average results are shown in Fig. 3.

There are three sub-figures in Fig. 3, which are the performances of the 10 methods related to m, n and β, respectively. We set the ordinate as the criterion value directly, because it can help us to evaluate the qualities of these methods more clearly. Moreover, since all the procedures are very fast ($O(n\log n)$ for the nine heuristics and $O(n)$ for the random one), we omit the running time of them. All the sub-figures indicate that $SPT\text{-}MinP$ is the best one among all the approaches, regardless of the aspects of m, n or β. Therefore, we can claim that $SPT\text{-}MinP$ is the most suitable one to get TS's initial solution, which makes our design of TS more reasonable.

Consequently, since there is no suitable lower bound or exact algorithms for the problem considered, we adopt the method used by Pesch and Sterna (2009) to analyze our algorithms, that is, the improvements from the (best) initial solution. More precisely, we use the value of

$$\frac{Y_{\text{init}} - Y_{\text{meta}}}{Y_{\text{init}}} \times 100\%$$

to report the qualities of the proposed meta-heuristic algorithms, where $init$ = $SPT\text{-}MinP$ since it is the best heuristics among the policies considered based on the above experiments, while $meta \in \{\text{TS, GA}\}$.

Table 6. Comparisons for parameter combinations in TS.

Combinations	Imp(%)	Time(ms)
TS_BBB	18.6074	9.6247
TS_BBS	18.6958	8.4296
TS_BSB	18.6139	8.1595
TS_BSS	18.6302	7.3931
TS_SBB	12.5759	4.8103
TS_SBS	12.3578	4.0259
TS_SSB	12.3802	3.6721
TS_SSS	12.4884	3.6086

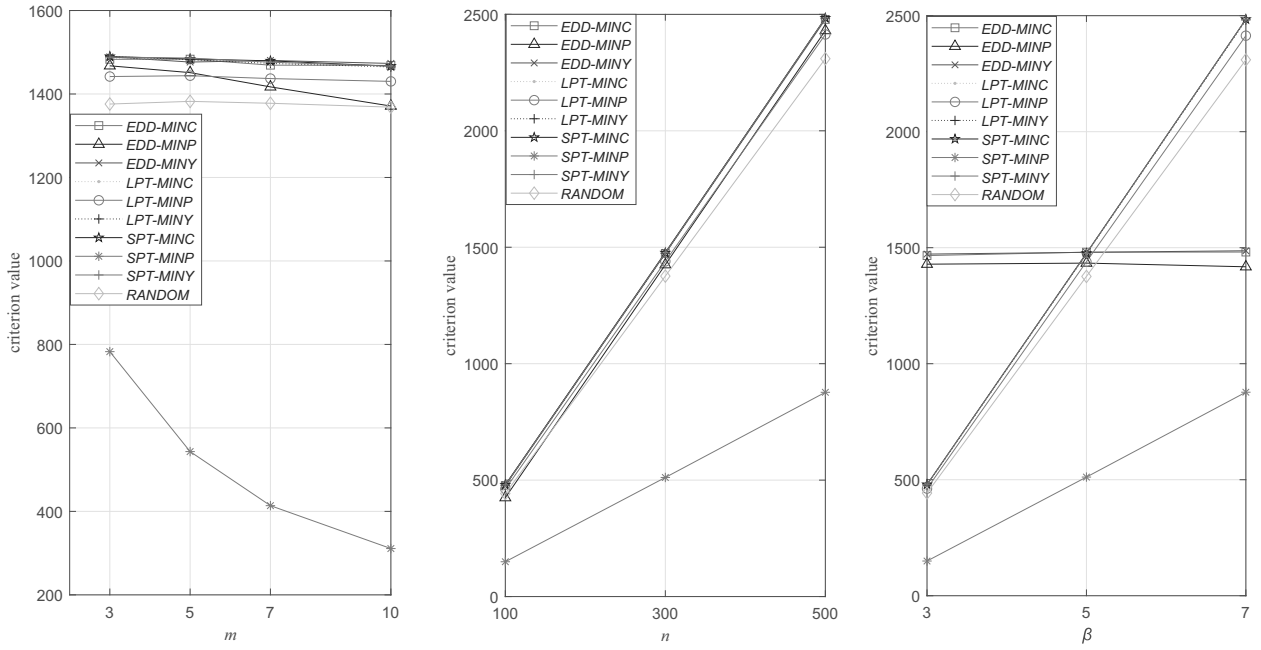
5.3. Parameter tuning for the meta-heuristics. The performance of any meta-heuristic algorithm is closely related to its parameter settings. In order to obtain the suitable parameters for TS and the GA, we introduce several supplementary experiments on parameter tuning in this part. We generated 20 random instances for each (m, n, β) based on the methods in Section 5.1, and adjusted the parameters of each meta-heuristic algorithm (i.e., size of neighborhood, long-term and short-term memories in TS, while population size, crossover and mutation probabilities in the GA).

5.3.1. Parameter tuning for TS. For each parameter in TS, i.e., size of neighborhood, long-term and short-term memories, we define a small value and a big value in the Table 5. Then we have 8 different parameter combinations for TS algorithm, denoted as TS_BBB (big neighborhood, big short-term memory, and big long-term memory), TS_BBS, TS_BSB, TS_BSS, TS_SBB, TS_SBS, TS_SSB, and TS_SSS, and their comparison results are shown in the Table 6. In this table, column "Imp(%)" represents the value of

$$\frac{Y_{\text{init}} - Y_{\text{meta}}}{Y_{\text{init}}} \times 100\%,$$

which can reflect the performance of the corresponding algorithm, while the columns "Time(ms)" reports the execution time in milliseconds.

Based on the results in Table 6, we claim that TS_BBS could produce the best solution among the eight parameter combinations. Although it takes more than



(a) Performances related to m. (b) Performances related to n. (c) Performances related to β.

Fig. 3. Comparisons among the simple heuristics.

Table 7. Parameter values in GA.

Parameter	Small value	Big value
Population size	50	200
Crossover probability p_c	0.8	0.95
Mutation probability p_m	0.05	0.2

twice the running time of the fastest one, i.e., TS_SSS, considering that the time unit is in milliseconds, we say that TS_BBS is still a very fast procedure. Therefore, we fix the parameters in our TS: the size of neighborhood is 40, and the sizes of short-term and long-term memories are 100 and 50, respectively (cf. Table 3).

5.3.2. Parameter tuning for the GA. In the same manner as in Section 5.3.1, we adjust the parameter values for the GA in this part. For its population size, crossover and mutation probabilities, we define their small and big values in Table 7, and report their comparison results in Table 8.

Therefore, we choose GA_BBB as our final setting for the GA, i.e., we set the population size to be 200, crossover and mutation probabilities to be 0.95 and 0.2, respectively (cf. Table 4).

5.4. Final evaluations on TS and the GA. Now we are ready to make the final evaluations on the proposed meta-heuristics in this paper, i.e., TS and the GA. We

Table 8. Comparisons for parameter combinations in the GA.

Combinations	Imp(%)	Time(ms)
GA_BBB	27.2035	387.3315
GA_BBS	20.1754	297.0859
GA_BSB	27.0946	376.2716
GA_BSS	20.2158	285.0183
GA_SBB	19.3598	81.2790
GA_SBS	10.5995	65.2739
GA_SSB	19.3727	79.0948
GA_SSS	10.7873	62.7952

further generated 50 instances for each combination of (m, n, β), and the average results are reported in Tables 9 and 10, respectively. In the same as in Tables 6 and 8, we use the columns "Imp(%)" to evaluate the main performances of the algorithms, and report their running time in the columns "Time(ms)" in milliseconds. Moreover, since we want to reveal the strengths of the algorithms more clearly, we count the numbers that one algorithm beats another within the 50 times experiments (on average), which are shown in the columns "Score".

Based on the experimental results, we could conclude the following characteristics of TS and the GA:

(i) Generally, the GA shows a better performance than TS. On one hand, the GA achieves a 26.9554% improvement over the initial solution (on the average, among 1800 instances), while the rate of TS is

Table 9. Performances of TS and the GA related to (m, n).

m	n	TS			GA		
		Imp(%)	Score	Time(ms)	Imp(%)	Score	Time(ms)
3	100	25.0713	16.67	5.7012	26.5639	33.33	105.4348
	300	18.1254	0.00	8.0770	24.7483	50.00	377.8662
	500	13.8957	0.00	10.9636	22.2385	50.00	636.4567
	$avg.$	19.0308	5.56	8.2473	24.5169	44.44	373.2526
5	100	26.7524	9.67	5.8147	30.2008	40.33	112.0703
	300	18.2465	0.00	8.1590	27.2803	50.00	381.7057
	500	13.8860	0.00	11.4578	23.8543	50.00	655.1272
	$avg.$	19.6283	3.22	8.4772	27.1118	46.78	382.9677
7	100	25.6073	6.00	5.9015	31.2694	44.00	112.7215
	300	17.5230	0.00	8.5882	28.3381	50.00	401.3114
	500	13.2418	0.00	10.9963	24.3097	50.00	635.1684
	$avg.$	18.7907	2.00	8.4953	27.9724	48.00	383.0671
10	100	22.9062	2.00	5.6853	33.0815	48.00	115.8920
	300	15.4456	0.00	8.3280	28.2093	50.00	387.6926
	500	11.6712	0.00	11.3368	23.3708	50.00	666.4315
	$avg.$	16.6743	0.67	8.4500	28.2206	49.33	390.0054
AVG.		18.5310	2.86	8.4175	26.9554	47.14	382.3232

Table 10. Performances of TS and the GA related to β.

β	TS			GA		
	Imp(%)	Score	Time(ms)	Imp(%)	Score	Time(ms)
3	22.5363	2.00	8.6888	34.0734	48.00	397.1892
5	18.0489	3.17	8.5976	25.7987	46.83	386.2916
7	15.0078	3.42	7.9660	20.9942	46.58	363.4888
AVG.	18.5310	2.86	8.4175	26.9554	47.14	382.3232

18.5310%. On the other hand, it wins on the average 47.14 times within 50 experiments (94.28%) for each parameter setting.

(ii) However, TS shows its fast process for the problem considered, with an average execution time 8.4175 ms, while this indicator of the GA is 382.3232 ms. Moreover, the running time of both the algorithms is not that sensitive to the parameters m and β, but sensitive to n, with an obvious reason of the encoding method.

(iii) Both the GA and TS are stable with the machines number m. From Table 9 or Fig. 4, we can find that for $m = 3$, 5, 7 and 10, the rates of the GA are almost around 26.96%, while the rates of TS are almost 18.53%.

(iv) For job n, both TS and the GA show strong influence along with its changing. As can be seen from Fig. 5, when $n = 100, 300$ and 500, the improvement rates of TS are on the average 25.0843%, 17.3351% and 13.1737%, while the ones for the GA are 30.2789%, 27.1440% and 23.4433%, both indicating a low performance when n turns big. Actually, this phenomenon shows in disguise that when the problem scale increases, the quality of the initial solution (obtained by SPT-$MinP$) is getting better.

(v) Finally, we can analyze the influence of β in Table 10 or in Fig. 6, where we find that both algorithms turn weak during an increase in β. Furthermore, the influence on the GA is much stronger than on TS, since the mean difference is about 6.54% in the GA for $\beta = 3, 5$ and 7; this value in TS is around 3.76%.

6. Conclusions

In this paper, we studied the problem of late work minimization on unrelated machines, i.e., $R||Y$, and proposed two meta-heuristic algorithms for it, which are tabu search (TS) and the genetic algorithm (GA), respectively. For TS, we first selected an initial solution among several heuristic schedules, and then defined its neighborhood exploration strategy and the tabu list. For the GA, the initial population was constructed by several heuristic and random solutions, and the iterations were

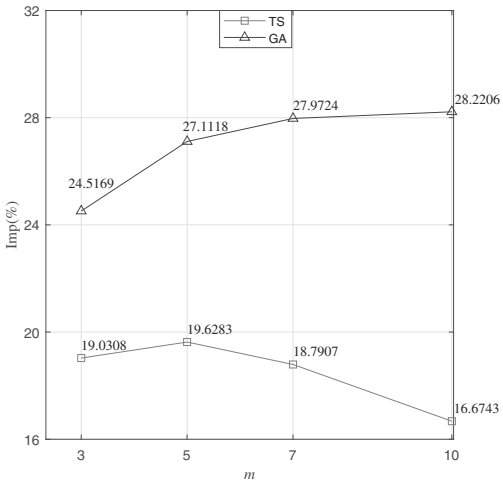


Fig. 4. Improvement rates related to m.

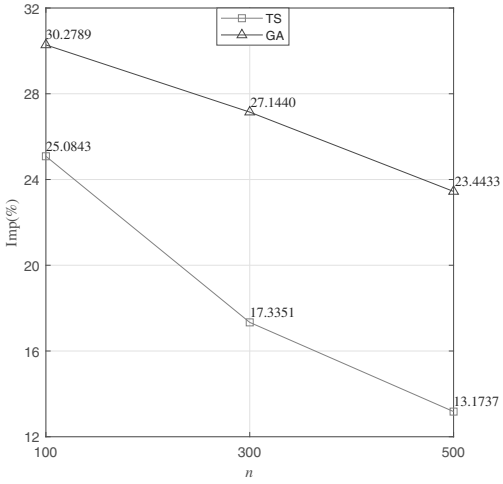


Fig. 5. Improvement rates related to n.

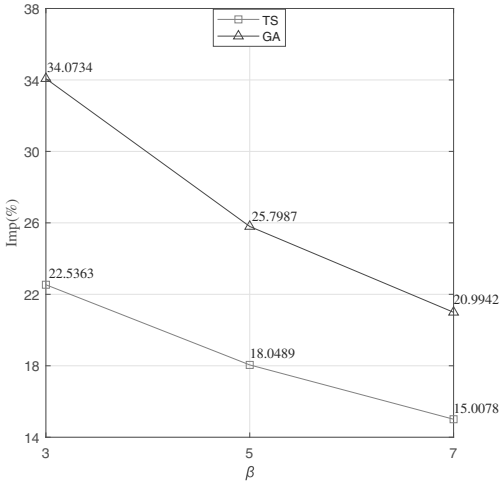


Fig. 6. Improvement rates related to β.

implemented through the operations such as selection, crossover, mutation, as well as elitist preservation. Moreover, we also defined the termination criteria for the two meta-heuristics to stop their evolutions and output the best-found solution.

In the phase of computational experiments, we first compared the qualities of 9 heuristic policies, and found that $SPT\text{-}MinP$ performed best among them. This result convinced us to use $SPT\text{-}MinP$ as the initial solution of TS, and as the reference for evaluating TS and the GA. Then we introduced the parameter tuning processes for TS and the GA, to find reasonable parameter setting, for them. During the major experiments for the two meta-heuristics, we found that the GA outperforms TS, but needs a longer time (but both of them are very fast since they all run in milliseconds). Finally, the impacts of problem parameters on the algorithms, such as m, n and β, were analyzed respectively.

To continue this work in the future, one may consider the lower bound analysis for the problem, which may lead to a deeper evaluation on the heuristic algorithms. Moreover, the multi-objective scheduling problems related to the due date constraint can be considered another future work, since it can model the practical productions more accurately.

Acknowledgment

This research was partially supported by the Overseas Training Foundation of Liaoning (no. 2019GJWYB015), the Natural Science Foundation of Liaoning (no. 2019-MS-170), a grant from the ICS PUT, and the China Scholarship Council (no. 201908210271).

References

Abasian, F., Ranjbar, M., Salari, M., Davari, M. and Khatami, S.M. (2014). Minimizing the total weighted late work in scheduling of identical parallel processors with communication delays, *Applied Mathematical Modelling* **38**(15): 3975–3986.

Błażewicz, J. (1984). Scheduling preemptible tasks on parallel processors with information loss, *Technique et Science Informatiques* **3**(6): 415–420.

Błażewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., Sterna, M. and Weglarz, J. (2019). *Handbook on Scheduling: From Theory to Practice*, Springer, Cham.

Błażewicz, J. and Finke, G. (1987). Minimizing mean weighted execution time loss on identical and uniform processors, *Information Processing Letters* **24**(4): 259–263.

Błażewicz, J., Kovalyov, M.Y., Musiał, J., Urbański, A.P. and Wojciechowski, A. (2010). Internet shopping optimization problem, *International Journal of Applied Mathematics and Computer Science* **20**(2): 385–390, DOI: 10.2478/v10006-010-0028-0.

Blazewicz, J., Pesch, E. and Sterna, M. (2000). The disjunctive graph machine representation of the job shop scheduling problem, *European Journal of Operational Research* **127**(2): 317–331.

Błażewicz, J., Pesch, E., Sterna, M. and Werner, F. (2004). Open shop scheduling problems with late work criteria, *Discrete Applied Mathematics* **134**(1): 1–24.

Błażewicz, J., Pesch, E., Sterna, M. and Werner, F. (2005). The two-machine flow-shop problem with weighted late work criterion and common due date, *European Journal of Operational Research* **165**(2): 408–415.

Błażewicz, J., Pesch, E., Sterna, M. and Werner, F. (2007). A note on the two machine job shop with the weighted late work criterion, *Journal of Scheduling* **10**(2): 87–95.

Blazewicz, J., Pesch, E., Sterna, M. and Werner, F. (2008). Metaheuristic approaches for the two-machine flow-shop problem with weighted late work criterion and common due date, *Computers & Operations Research* **35**(2): 574–599.

Chen, R., Yuan, J., Ng, C. and Cheng, T. (2019). Single-machine scheduling with deadlines to minimize the total weighted late work, *Naval Research Logistics (NRL)* **66**(7): 582–595.

Chen, X., Chau, V., Xie, P., Sterna, M. and Błażewicz, J. (2017). Complexity of late work minimization in flow shop systems and a particle swarm optimization algorithm for learning effect, *Computers & Industrial Engineering* **111**: 176–182.

Chen, X., Kovalev, S., Sterna, M. and Błażewicz, J. (2020a). Mirror scheduling problems with early work and late work criteria, *Journal of Scheduling*, DOI:10.1007/s10951-020-00636-9.

Chen, X., Liang, Y., Sterna, M., Wang, W. and Błażewicz, J. (2020b). Fully polynomial time approximation scheme to maximize early work on parallel machines with common due date, *European Journal of Operational Research* **284**(1): 67–74.

Chen, X., Sterna, M., Han, X. and Błażewicz, J. (2016). Scheduling on parallel identical machines with late work criterion: Offline and online cases, *Journal of Scheduling* **19**(6): 729–736.

Gerstl, E., Mor, B. and Mosheiov, G. (2019). Scheduling on a proportionate flowshop to minimise total late work, *International Journal of Production Research* **57**(2): 531–543.

Glover, F. (1989). Tabu search. Part I, *ORSA Journal on Computing* **1**(3): 190–206.

Glover, F. (1990). Tabu search. Part II, *ORSA Journal on Computing* **2**(1): 4–32.

Graham, R., Lawler, E., Lenstra, J. and Kan, A.R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey, *Annals of Discrete Mathematics* **5**: 287–326.

Hariri, A.M., Potts, C.N. and Van Wassenhove, L.N. (1995). Single machine scheduling to minimize total weighted late work, *ORSA Journal on Computing* **7**(2): 232–242.

Holland, J.H. (1962). Outline for a logical theory of adaptive systems, *Journal of the ACM* **9**(3): 297–314.

Kovalyov, M.Y., Potts, C.N. and Van Wassenhove, L.N. (1994). A fully polynomial approximation scheme for scheduling a single machine to minimize total weighted late work, *Mathematics of Operations Research* **19**(1): 86–93.

Kundakcı, N. and Kulak, O. (2016). Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem, *Computers & Industrial Engineering* **96**: 31–51.

Labib, N.S., Danoy, G., Musial, J., Brust, M.R. and Bouvry, P. (2019). Internet of unmanned aerial vehicles—A multilayer low-altitude airspace model for distributed UAV traffic management, *Sensors* **19**(21): 4779.

Leung, J.Y. (2004). Minimizing total weighted error for imprecise computation tasks and related problems, *in* J.Y. Leung (Ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapman and Hall/CRC, Boca Raton, FL.

Lin, B.M. and Hsu, S. (2005). Minimizing total late work on a single machine with release and due dates, *SIAM Conference on Computational Science and Engineering, Orlando, FL, USA*.

Lin, B.M., Lin, F. and Lee, R. (2006). Two-machine flow-shop scheduling to minimize total late work, *Engineering Optimization* **38**(04): 501–509.

Lopez-Loces, M.C., Musial, J., Pecero, J.E., Fraire-Huacuja, H.J., Blazewicz, J. and Bouvry, P. (2016). Exact and heuristic approaches to solve the Internet shopping optimization problem with delivery costs, *International Journal of Applied Mathematics and Computer Science* **26**(2): 391–406, DOI: 10.1515/amcs-2016-0028.

Pesch, E. and Sterna, M. (2009). Late work minimization in flow shops by a genetic algorithm, *Computers & Industrial Engineering* **57**(4): 1202–1209.

Piroozfard, H., Wong, K.Y. and Wong, W.P. (2018). Minimizing total carbon footprint and total late work criterion in flexible job shop scheduling by using an improved multi-objective genetic algorithm, *Resources, Conservation and Recycling* **128**: 267–283.

Potts, C.N. and Van Wassenhove, L.N. (1992a). Approximation algorithms for scheduling a single machine to minimize total late work, *Operations Research Letters* **11**(5): 261–266.

Potts, C.N. and Van Wassenhove, L.N. (1992b). Single machine scheduling to minimize total late work, *Operations Research* **40**(3): 586–595.

Rybarczyk, A., Hertz, A., Kasprzak, M. and Blazewicz, J. (2017). Tabu search for the RNA partial degradation problem, *International Journal of Applied Mathematics and Computer Science* **27**(2): 401–415, DOI: 10.1515/amcs-2017-0028.

Servranckx, T. and Vanhoucke, M. (2019). A tabu search procedure for the resource-constrained project scheduling problem with alternative subgraphs, *European Journal of Operational Research* **273**(3): 841–860.

Sterna, M. (2007). Metaheuristics for late work minimization in two-machine flow shop with common due date, *Computers & Industrial Engineering* **52**(2): 210–228.

Sterna, M. (2011). A survey of scheduling problems with late work criteria, *Omega* **39**(2): 120–129.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*, Wiley & Sons, Hoboken, NJ.

Whitley, D. (1994). A genetic algorithm tutorial, *Statistics and Computing* **4**(2): 65–85.

Wu, C.-C., Yin, Y., Wu, W.-H., Chen, H.-M. and Cheng, S.-R. (2016). Using a branch-and-bound and a genetic algorithm for a single-machine total late work scheduling problem, *Soft Computing* **20**(4): 1329–1339.

Yan, F., Dridi, M. and El Moudni, A. (2013). An autonomous vehicle sequencing problem at intersections, *International Journal of Applied Mathematics and Computer Science* **23**(1): 183–200, DOI: 10.2478/amcs-2013-0015.

Wen Wang is currently a dual-degree (master) student both at the Liaoning University of Technology and the Poznań University of Technology. Her research interests include intelligent optimization algorithms and their applications, especially for scheduling problems.

Xin Chen holds a BE in computer science from the Dalian University of Technology, China (2005), and a PhD in information science from the Poznań University of Technology, Poland (2014). He is currently an associate professor at the School of Electronic and Information Engineering of the Liaoning University of Technology, China. His research interests include combinatorial optimization, especially scheduling problems and algorithm design.

Jedrzej Musial is an assistant professor at the Poznań University of Technology. His research interests include e-commerce, Internet shopping, cloud brokering, algorithms, applications of combinatorial optimization and operations research. He has a PhD and habilitation (DSc) in computer science from the Poznań University of Technology and a PhD from the University of Luxembourg. He has published 35 papers and 2 monographs.

Jacek Blazewicz is a full professor at the Poznań University of Technology (PUT) and the director of the Institute of Computing Science there. His research interests include algorithm design, computational complexity, scheduling, combinatorial optimization, bioinformatics, e-commerce. He has a PhD and habilitation (DSc) in computer science from the Poznań University of Technology. His publication record includes over 400 papers in many outstanding journals. He is also the author and a co-author of over ten monographs. His Web of Science citation index is over 4050 and his h-index is 30. He is an IEEE fellow.

Received: 20 February 2020
Revised: 6 June 2020
Accepted: 2 July 2020