Arabian Journal for Science and Engineering (2022) 47:3899–3913
https://doi.org/10.1007/s13369-021-06244-9

RESEARCH ARTICLE-SYSTEMS ENGINEERING

Effective Heuristics and an Iterated Greedy Algorithm to Schedule Identical Parallel Machines Subject to Common Restrictive Due Windows

Gustavo Alencar Rolim[1] · **Marcelo Seido Nagano**[1] · **Bruno de Athayde Prata**[2]

Received: 24 May 2021 / Accepted: 20 September 2021 / Published online: 14 October 2021

Abstract
In this paper, we address a variant of the identical parallel machines scheduling problem subject to common restrictive due windows. The performance measure adopted is the minimization of total weighted earliness and tardiness. Since the variant under study is an NP-hard problem for two or more machines, we develop a family of constructive heuristics, which are comprised of four phases. First, jobs are sequenced according to priority rules. Second, jobs are assigned to machines using a greedy strategy. Third, a local search is performed to find a better distribution of jobs into machines. Fourth, two heuristics are applied for individually sequencing jobs in each machine, namely RN-RGH and RN-SEA. In addition, we also propose an iterated greedy algorithm to improve the solutions of the best performing heuristic. The computational experiments were carried out to prove the ability of these heuristics to find high-quality solutions in acceptable CPU time. More specifically, the RN-SEA family of algorithms stands out as the most efficient for the problem, however, with a higher computational effort. We also confirm that the IG algorithm has the potential for improving existing solutions, specially for problems with two machines and instances with up to 100 jobs in size.

Keywords Machine scheduling · Earliness and tardiness · Identical parallel machines · Common due window · Heuristics · Iterated greedy

1 Introduction

Industrial engineering practitioners have been devoting a lot of attention to production scheduling problems. Usually, the objective functions considered in this type of problem involve nondecreasing functions of job completion times. However, these performance measures do not recognize the costs related to the early jobs. With the propagation of the just-in-time (JIT) management system, the adoption of objective functions considering both earliness and tardiness (ET) has become a reality for production planners [1].

Commonly, due dates may be treated as decision variables or given constraints. In the first case, they reflect production targets that guide the progress of internal tasks. For different models involving due date determination decisions we refer to [2–4]. In contrast, the second case refers to frequent situations where shop floor activities must meet customer demands which are influenced by external factors [5].

In many practical contexts, if a customer order goods from a supplier, a fixed delivery date is agreed upon. Frequently, the customer accepts a small deviation from this delivery date as there is an uncertainty degree on the supplier's side. This uncertainty may arise in different ways, such as machine breakdowns, resource availability, defects in raw materials, and other production issues [6]. With that said, it may be suitable to negotiate a common due window (CDW).

Examples of such production environments include assembly lines that require multiple components to build a final product or situations where several jobs constitute a single customer order. These contexts often arise in JIT manufac-

✉ Marcelo Seido Nagano
drnagano@usp.br

Gustavo Alencar Rolim
gustavo.rolim@usp.br

Bruno de Athayde Prata
baprata@ufc.br

1 Department of Production Engineering, São Carlos School of Engineering, University of São Paulo, São Carlos, Brazil

2 Department of Industrial Engineering, Federal University of Ceará, Campus do Pici, Fortaleza, Brazil

turing, chemical processing and PERT/CPM scheduling, and information technology [7–9]. Moreover, there are applications in modern semiconductor fabrication facilities where a large number of chips may be subject to the same due window. For more details regarding semiconductor facilities, we refer to [10–12].

It is also important to note that production control strategies based on dispatching rules are popular strategies adopted by many manufacturing industries. However, the increasing degree of automation attracted the attention of scheduling approaches from researchers and people from industry for the last few decades. This fact occurred due to the intensification of automated real-time data collection as an alternative to manual production control that often leads to an imprecise behavior of complex manufacturing systems. This phenomenon is one of the guides of the new paradigm of Industry 4.0 specially because optimization is one of its key enabling technologies [11,13].

The contributions related to the identical parallel machine scheduling problem (IPMSP) considering a CDW are rather limited, despite the theoretical and practical importance of this variant. It was not possible to observe solution procedures developed to solve test instances with 40 or more jobs. [14] propose a branch-and-bound (B&B) procedure with a column generation algorithm where the problem is formulated as a set partitioning-type model and. Also, in each B&B iteration, the linear relaxation of this formulation is solved by the standard column generation procedure. With that said, the main contributions of this paper are listed as follows:

- To the best of our knowledge, this is the first study to provide simple and effective heuristics for the IPMSP with a CDW. It not only fills this gap, but also provides a practical approach to solve the problem.
- We propose a method based upon five different priority rules. First, the initial job list is ordered according to some predefined criteria and then the jobs are assigned to the machines following a greedy strategy.
- Once an initial solution is constructed, a local search procedure is applied to find a better distribution of jobs into machines.
- Afterward, we also propose two different heuristics to solve each machine subproblem separately.
- Finally, an iterated greedy (IG) algorithm is used to improve the solution of the best performing heuristic.

The remainder of this paper is divided as follows. Section 2 presents a literature review of parallel machine scheduling problems with CDW considerations. We also present some recent advances in CDW research. Section 3 defines the problem and its structural properties. Then, in Sect. 4, we describe the proposed priority rules, local search, and iterated greedy algorithm. We carry a discussion on the experimental results in Sect. 5, and finally, Sect. 6 presents the concluding remarks as well as directions for further research.

2 Literature Review

To the best of our knowledge, [15] were the first to provide an extension from the single-machine environment to multiple parallel machines when the due window is not restricted (the left boundary of the window may be greater than the sum of processing times) and the ET weights are job independent and asymmetric. They show that this problem is NP-hard even for the unit weight case and provide a pseudopolynomial-time dynamic programming algorithm for the 2-machine case as well as a heuristic with absolute error bound. The heuristic is extended for an arbitrary number of machines. The error tends to decrease if the number of jobs is reasonably large.

[14] provide a branch-and-bound algorithm for the case when the due window is restrictive and the weights are job-dependent. Their algorithm is based on the column generation approach in which the problem is first formulated as a set partitioning-type formulation, and then, in each branch-and-bound iteration, the linear relaxation of this formulation is solved by the standard column generation procedure. Results are reported for instances with up to 40 jobs.

[16] consider the due window assignment problem with unit processing times in identical parallel machines. They show that this problem is solvable in $O(1)$ time and provide several properties of an optimal solution. In a similar problem, but with job-dependent ET weights and an additional penalty for due window size, [17] give an efficient solution procedure that runs in $O(n^4)$. [18] extended this problem by adding a penalty for the left boundary of the due window and established conditions for optimal schedules. Other special cases that require less computational efforts are described. [19] shows that the general problem discussed by [18] can be solved by a lower order algorithm in $O(n^3)$ with some improvements.

The case of uniform parallel machines with identical processing times is reported by [20] where they show that the two-machine environment may be solved in constant time. [21] assume that the due window can be either restrictive or nonrestrictive. It was also assumed that the ET weights are job dependent and the jobs share an identical unit processing time. With that said, they propose $O(3^m n^3)$ and $O(2^m n^3)$ algorithms for restrictive and nonrestrictive settings, respectively.

[22] address the problem of due window assignment and scheduling when jobs have independent earliness and tardiness costs, and penalties are also incurred for due window size and location. They establish a number of properties of optimal solutions and derive dynamic programming algorithms,

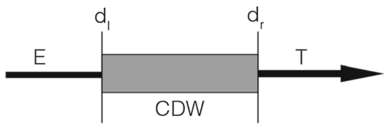


Fig. 1 Common due window representation

which are pseudopolynomial if the number of machines is considered to be constant.

3 Problem Statement, Definitions, and Dominance Properties

To a certain extent, the problem under study is similar to the one reported by [14]. This problem consists of a set of n independent jobs, $N = \{1, 2, ..., n\}$, that need to be processed independently on any of the m ($m > 1$) identical parallel machines. The objective is to determine the sequence of all jobs so as to minimize the total weighted earliness and tardiness penalties. Using the well-known three field classification scheme introduced by [23] and adapted by [5] for scheduling problems with common due windows, the IPMSP can be expressed according to the notation $P|\langle\hat{d}_l, \hat{d}_r\rangle|\sum(\alpha_j E_j + \beta_j T_j)$, where $\langle\hat{d}_l, \hat{d}_r\rangle$ designates a common restrictive due window, that is, its left boundary must satisfy $d_l < \sum_{j=1}^{n} p_j$, E_j and T_j denote the earliness and tardiness of job $j \in N$, and α_j and β_j are job-dependent weights, respectively. A graphical representation of the due window is given in Fig. 1.

After being processed on the machine, and as a result of scheduling decisions, job j will be assigned a completion time denoted C_j. In other terms, the earliness and tardiness can be expressed as:

$$E_j = \max(d_l - C_j, 0) = (d_l - C_j)^+ \tag{1}$$

$$T_j = \max(C_j - d_r, 0) = (C_j - d_r)^+ \tag{2}$$

It is important to notice that, in contrast with tardiness penalties, earliness penalties are nonincreasing in C_j which makes the objective function a nonregular performance measure. For the common due window, let d_l and d_r be integers indicating the earliest and the latest due dates, respectively. Then, its size and location may be given as Eqs. 3 and 4:

$$d_l = \lfloor h_l \times (\frac{\sum_{j=1}^{n} p_j}{m}) \rfloor \tag{3}$$

$$d_r = \lfloor h_r \times (\frac{\sum_{j=1}^{n} p_j}{m}) \rfloor \tag{4}$$

Here, h_l and h_r are given parameters for defining d_l and d_r, respectively. While the parameters h_l and h_r could be specified by any value, it is important to establish their ranges as $1 > h_r > h_l > 0$ so that the common due window becomes restricted by $0 < d_l < d_r < (\sum_{j=1}^{n} p_j/m)$. This configuration allows to avoid some trivial cases, for example, when $h_l = 0$ and $h_r = 1$ the common due window size would be, at least, close enough to $(\sum_{j=1}^{n} p_j/m)$; therefore, it could be possible to fit all jobs in the common due window. Moreover, if $h_l = h_r$, then the common due window could be reduced to a time instant; hence, no tolerance interval would exist; instead, it would be considered as a common due date. Other important assumptions are given as follows:

- All jobs are independent and ready at time zero.
- The machines can process only one job at a time.
- Jobs are processed without any interruptions, that is, machine breakdowns, planned or unexpected maintenance, and other effects that cause disruptions are not considered.
- The processing time of a given job is constant and cannot be compressed or expanded.
- No preemptions are permitted.

It is important to notice that the problem under study has some special characteristics that constitute optimal solutions. In a simplified way, these structural properties are given as follows:

- **Property 1**: There are no inserted idle times between jobs.
- **Property 2**: Jobs finishing before d_l are sequenced in nonincreasing order of the ratio p_j/α_j.
- **Property 3**: Jobs starting and finishing after d_r are sequenced in nondecreasing order of the ration p_j/β_j.
- **Property 4**: Jobs starting and finishing in the window can be sequenced in an arbitrary order.
- **Property 5**: In each machine, there exists an optimal schedule in which either the processing of the first job starts at time zero or one job is completed at d_l or d_r.

The proofs of these dominance rules can be found in [14]. It is clear from properties 2, 3, and 4 that an optimal schedule follows the V-shaped property in each machine. Hereafter, the notations that will be used throughout the paper are established in Table 1.

It turns out that optimal schedules may have several different configurations. For this reason, we adopt the mapping procedure in [6] to guide the process of constructing a heuristic sequence. Despite the fact that an optimal schedule is V-shaped in each machine, it is essential to point out that it may contain one or two straddling jobs that are not necessarily subject to this property. We define a left-straddling job s_l if it straddles d_l, and right-straddling job s_r if it straddles d_r. Moreover, it is possible the existence of an optimal schedule with a double-straddling job s_d. In other words, a job com-

Table 1 Notations

Notation	Description
d_l	Left border of the common due window
d_r	Right border of the common due window
n	Number of jobs
m	Number of machines
j, k, l	Index of a job
i	Index of a machine
p_j	Processing time of job j
α_j	Earliness penalty of job j per time unit
β_j	Tardiness penalty of job j per time unit
E_j	Earliness of job j
T_j	Tardiness of job j
$s_{[l]}$	Leading idle time, i.e., starting time of the first job
C_j	Completion time of job j
S_j	Starting time of job j
J_E	Set of no-tardy jobs scheduled to finish before or in d_l, $J_E = \{j \| C_j \leq d_l\}$
J_T	Set of tardy jobs scheduled to begin after or in d_r, $J_T = \{j \| C_j - p_j \geq d_r\}$
J_W	Set of jobs that start and finish in the due window, $J_W = \{j \| C_j - p_j \geq d_l \vee C_j \leq d_r\}$
s_l	Left-straddling job, $S_{s_l} < d_l \vee C_{s_l} > d_l$
s_r	Right-straddling job, $S_{s_r} < d_r \vee C_{s_r} > d_r$
s_d	Double-straddling job, $S_{s_d} < d_l \vee C_{s_d} > d_r$

pletely overlaps the common due window. From property 5, it also becomes evident that an optimal schedule might exist in which the first job does not start at time zero. We should call the time span before the starting of the first job as leading idle time.

4 Proposed Solution Approaches

4.1 Priority Rules and Greedy Heuristic

Any algorithm designed for solving the PMSP with ET penalties subject to common due windows has two main decisions to address. The first one is how to assign jobs to machines, while the second concerns to the sequencing aspects on each individual machine. In this section, we establish a procedure devoted to find an initial schedule on each machine.

Rather than assigning each job to a machine without any predefined criteria, we decided to sort the jobs according to five different priority rules. These rules were established by [24] for a similar PMSP problem with common due date. They tested several other rules; however, the following ones presented the best results:

- **Rule 1**: Nonincreasing $\beta_j p_j$.
- **Rule 2**: Nonincreasing $\beta_j p_j/\alpha_j$.
- **Rule 3**: Nonincreasing p_j.
- **Rule 4**: Nonincreasing β_j.
- **Rule 5**: Nondecreasing p_j/β_j.

Once the whole list of jobs is sorted, the jobs are assigned to each machine using a greedy heuristic. The first $m - 1$ jobs are distributed into the first $m - 1$ machines. Thereafter, the algorithm proceeds by promoting a competition between the next two jobs in the list. Calculations are made to determine in which machine and position the next two jobs attain a minimum cost increase. Then, the one that produces the minimum cost is assigned to the best position and the other is returned to the head of the list. The algorithm stops when the whole list is empty and the jobs are fully assigned to the machines. Algorithm 1 shows the pseudocode of the greedy heuristic.

4.2 Local Search

The local search (LS) procedure was designed to exchange jobs between two machines that have different job charges. In other words, since the initial assignment does not consider the balance between earliness and tardiness penalties as well as processing times, LS looks for a decrease in the cost function by moving jobs from a machine to another. The following steps summarize the underlying idea of the LS, and Algorithm 2 presents the pseudocode.

Algorithm 1: Greedy heuristic

```
Input: Initial job list ($N$)
Output: A schedule ($\pi$)
$N \leftarrow \{1, 2, ..., n\}$, $M \leftarrow \{M_1, M_2, ..., M_m\}$, $flag \leftarrow true$
Sort $N$ according to one of the priority rules
while $N \neq \emptyset$ do
    if $flag = true$ then
        for $i = 1$ to $m - 1$ do
            Let job in position $[j]$ be the first job in $N$
            Assign job in position $[j]$ to $M_i$
            $[j] \leftarrow [j + 1]$
        end
        $flag \leftarrow false$
    else
        for $i$ = 1 to m do
            Insert jobs $[j]$ and $[j + 1]$ in every position of $M_i$
            Save the best position in case of a minimum cost increase
        end
        Assign the job that produced the minimum cost increase to the best machine position
        Update the first job of $N$
    end
end
```

- Sort the machines in nonincreasing order of costs.
- Compute the total processing time (D_i^T), the sum of earliness penalties (α_i^T), and the sum of tardiness (β_i^T) penalties of each machine.
- Try to move all jobs $j \in M_i$ to all possible positions of M_{i+1} with a lower cost if at least one of the conditions holds: $D_i^T > D_{i+1}^T + p_j \vee \alpha_i^T > \alpha_{i+1}^T + \alpha_j \vee \beta_i^T > \beta_{i+1}^T + \beta_j$.
- If an improvement in the total cost is achieved, move job j from M_i to M_{i+1}.
- Repeat until all machines are tested.

4.3 Solving the Single-Machine Subproblems

Once the jobs are distributed to machines, it is necessary to find better sequences on each machine. For this, two strategies are adopted: the RGH heuristic and SEA. Both deterministic methods are used to order the sequences on each machine. This is the third phase of the solution procedure. Hereafter, the nomenclature adopted to differentiate each solution procedure is RN-RGH and RN-SEA. Note that the nomenclature is followed by a number from 1 to 5, which distinguishes the 5 priority rules. The next two subsections describe these solutions procedures in detail.

4.3.1 RN-RGH

The RN-RGH algorithm is based on the procedure developed by [25] for the single-machine problem with a common due

Algorithm 2: Local Search

```
Input: Initial schedule ($\pi$)
Output: An improved schedule ($\pi^{new}$)
$M \leftarrow \{M_1, M_2, ..., M_m\}$, $flag \leftarrow true$
Sort the machines in nonincreasing order of costs
for i = 1 to m - 1 do
    while flag = true do
        Compute $D_i^T, \alpha_i^T, \beta_i^T, D_{i+1}^T, \alpha_{i+1}^T, \beta_{i+1}^T$
        for all jobs $j \in M_i$ do
            if $D_i^T > D_{i+1}^T + p_j \vee \alpha_i^T > \alpha_{i+1}^T + \alpha_j \vee \beta_i^T > \beta_{i+1}^T + \beta_j$ then
                Try to move job $j$ from $M_i$ to $M_{i+1}$ by inserting it in every possible position
                Compute $f(\pi^{new})$
                if $f(\pi^{new}) < f(\pi)$ then
                    Move job $j$ to the best position in $M_{i+1}$
                    $flag \leftarrow true$
                else
                    Keep $\pi$ unchanged
                    $flag \leftarrow false$
                end
            else
                $flag \leftarrow false$
            end
        end
    end
end
```

window. The basic idea is to employ the constructive heuristic reported by these authors as an improvement method.

In a rather simplified manner, once the jobs are better distributed into the machines, their heuristic is applied in each machine separately. The algorithm starts by assigning the jobs that have high earliness and tardiness weights relative to their processing times to set J_W. Afterward, from the remaining jobs, those which possess a high tardiness weight relative to the earliness weight are assigned to set J_E. Here, we omit the details of the algorithm due to space restrictions. For details and a numerical example we refer to [25]. Again, their algorithm will be repeated in each machine and a new schedule will be constructed.

4.3.2 RN-SEA

The other improvement strategy is based on a Sequential Exchange Approach [26]. Here, a new heuristic is shown to schedule each machine against a common due window. Since there are three sets of jobs that constitute a solution for the problem (J_E, J_W, J_T), RN-SEA swaps or moves jobs between these sets systematically to derive a near-optimal solution. For the sake of simplicity, only two sets are considered since jobs in J_W may be sequenced in an arbitrary order. Now, let $J_{E'}$ denote the set of no-tardy jobs including the ones that start and finish in the due window ($J_{E'} = J_E \cup J_W$). Possible straddling jobs (s_l, s_r, and s_d) are also in $J_{E'}$ since they start processing before the right boundary of the due window.

In the forward exchange step, for each available machine, the jobs in $J_{E'}$ are chosen individually from the sequence obtained after phases one and two. Then, this job $j \in J_{E'}$ is matched with all jobs in J_T in order to verify the swap which would lead to the largest improvement of the cost function value. Moreover, we can insert a job $k \in J_{E'}$ in the set J_T if a better value for the objective function can be reached. If the swap move incurs an improvement of the objective-function value, this movement is performed. After evaluating both moves, the best movement is performed, and the jobs are ordered according to the V-shaped property again.

After that, the backward exchange operator is performed. In this step, jobs in J_T are selected individually considering the sequence found in the previous step, and jobs are swapped with jobs in $J_{E'}$ to determine the best improvement of the objective function value. Finally, the movement with the best improvement is then selected, and the sequence is constructed also considering the V-shaped property.

Since the common due windows are restricted in relation to the total processing time, the start time of the first job to be scheduled is set to be zero. Evidently, for instances where the left boundary (d_l) has a larger restrictive factor (h_l), it may be the case that possible straddling jobs should be eliminated to obtain better objective function values; therefore, the procedure identifies them and produces backward-shift solutions of right and left straddling jobs. The start time is calculated as $s_{[l]} = d_l - S_{s_l}$ and $s_{[l]} = d_r - S_{s_r}$. Of course, jobs should be rearranged according to the V-shaped property and the solution with the best objective function value is kept.

4.4 Iterated Greedy

To put it briefly, IG generates a sequence of solutions by iterating over greedy constructive heuristics using two important phases: destruction and construction. During the destruction phase, some solution components are removed from a previously constructed complete candidate solution. The construction procedure then applies a greedy constructive heuristic to reconstruct a complete candidate solution. Once this new solution is constructed, an acceptance criterion is used to select a new incumbent solution. The algorithm iterates over these steps until a stopping condition is reached. Note that, IG is closed related to iterated local search (ILS); however, instead of iterating over a local search, it iterates over construction heuristics [27]. Figure 2 depicts a hypothetical iteration of the IG algorithm in one of the machines, and Algorithm 3 outlines the step-by-step procedure.

Algorithm 3: Iterated greedy

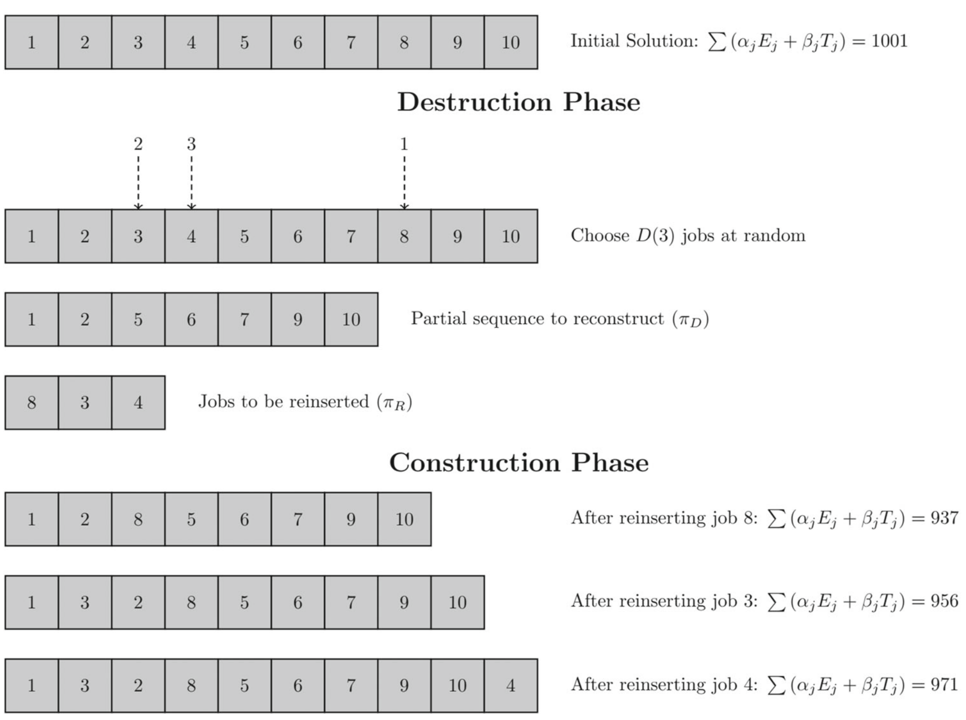
```
1  Input: Initial schedule ($\pi$)
2  Output: An improved schedule ($\pi^{best}$)
3  while stop condition is not satisfied do
                                        // Destruction phase
4
5      $\pi' \leftarrow \pi$
6      $\pi^{best} \leftarrow \pi$
7      for $j = 1$ to $D$ do
8          $\pi' \leftarrow$ Remove one job at random from $\pi'$ and insert
           in $\pi'_R$
9      end
                                        // Construction phase
10     for $j = 1$ to $D$ do
11         $\pi' \leftarrow$ best permutation obtained by inserting job
           $j \in \pi'_R$ in all possible positions of $\pi'$ according to the
           V-shaped property
12     end
                                        // Local search
13     $\pi'' \leftarrow Local\_search(\pi')$
14     if $f(\pi'') < f(\pi)$ then
15         $\pi \leftarrow \pi''$
16         if $f(\pi'') < f(\pi^{best})$ then
17             $\pi^{best} \leftarrow \pi$
18         end
19     end
                                        // Acceptance criterion
20     if $Rand \leq exp\{-f(\pi'') - f(\pi)/Temp\}$ then
21         $\pi \leftarrow \pi''$
22     end
23 end
```

4.4.1 Destruction and Construction

Two central procedures in any IG algorithm are the destruction and construction phases. The destruction phase is applied to a permutation π of n jobs, and it chooses randomly and without repetition D jobs. These jobs are then removed from π in the order in which they were chosen. As a result, there are two subsequences, the first is a partial sequence π_D with $n - D$ jobs and another sequence of D jobs which is denoted as π_R. It is worth noting that π_R contains the jobs that need to be reinserted into π_D in the order in which they were removed from π. Here, the destruction and construction procedures are repeated in every machine to build a new feasible solution. Once a complete feasible solution is generated, the LS procedure described in Algorithm 2 is applied.

4.4.2 Acceptance Criterion

After the local search phase, an acceptance criterion is applied to determine whether the new generated solution π' is accepted or not as the incumbent solution π for the next iteration. The simplest way to establish an acceptance criterion is to accept only better solutions. However, an IG algorithm using such strategy may lead to premature convergence due



1 2 3 4 5 6 7 8 9 10
Initial Solution: $\sum(\alpha_j E_j + \beta_j T_j) = 1001$
Destruction Phase
2 3 1
1 2 3 4 5 6 7 8 9 10
Choose $D(3)$ jobs at random
1 2 5 6 7 9 10
Partial sequence to reconstruct (π_D)
8 3 4
Jobs to be reinserted (π_R)
Construction Phase
1 2 8 5 6 7 9 10
After reinserting job 8: $\sum(\alpha_j E_j + \beta_j T_j) = 937$
1 3 2 8 5 6 7 9 10
After reinserting job 3: $\sum(\alpha_j E_j + \beta_j T_j) = 956$
1 3 2 8 5 6 7 9 10 4
After reinserting job 4: $\sum(\alpha_j E_j + \beta_j T_j) = 971$

Fig. 2 Example of one iteration of the proposed IG heuristic

to insufficient diversification [27]. Other authors prefer to choose a fixed probability of accepting worst solutions, which is a parameter to be determined [28,29]. Other authors opt for a simulated annealing acceptance criterion where the acceptance probability is based on a control parameter which may be a constant [27,30,31] or a variable temperature [32,33]. In this study, we adopt the simulated annealing acceptance criterion described in [27] with a constant temperature. The notation *Rand* of the acceptance criterion depicted in Algorithm 3 refers to a random generated number taken from the interval [0, 1].

4.4.3 Stopping Conditions

The stopping condition commonly involves a fixed number of iterations, a depleted execution time, or the detection of algorithmic stagnation. In this study, only one stopping condition is used to force the IG algorithm to be terminated, which consists of the depleted execution time. In other words, once a certain amount of CPU time is reached, the IG algorithm is stopped.

4.4.4 Parameter Settings

The proposed IG presents the following parameters:

- The number of jobs to be removed for a machine (D).
- The temperature ($Temp$), a control parameter presented in the simulated annealing algorithms. The probability to accept movements worsening the objective-function values decreases progressively with the reduction of the temperature.
- The stopping criterion (t).

The parameter D is given by $D = \lceil d' * n_m \rceil$, in which $0 < d' < 1$ and n_m is the number of jobs assigned to a given machine. We define the stop condition as the maximal computational time, given by $t * n$ seconds ($0 < t < 1$). Since the proposed IG can be executed with distinct parameter values, we calibrate these values using an experimental design approach.

To avoid the risks of over-fitted results, it was selected 5 out of 10 instances of each size totally at random. For this, 5 nonrepeated numbers between 0 and 10 were generated and the respective instances were taken from the test set, e.g., each of the 10 instances of size $n = 20$ were matched with random indexes and taken as samples to determine the parameter levels. This way a total of $5 \times 3 \times 5 \times 5 = 375$ instances were solved for different parameter configurations. With a fixed CPU time equal to $0.10 * n$ seconds ($t = 0.10$), a full factorial experiment with d' and $Temp$ was conducted and each parameter is tested at four levels resulting in a total of 16 different algorithms. Each configuration is reported in Table 2.

After setting each version of the IG algorithm to solve each instance 5 times, the quality of the solutions is measured by means of the RPD, which is computed according to the following equation:

Table 2 Full factorial experiment

Algorithm	d'	$Temp$
Alg 1	0.01	5
Alg 2	0.01	10
Alg 3	0.01	20
Alg 4	0.01	50
Alg 5	0.05	5
Alg 6	0.05	10
Alg 7	0.05	20
Alg 8	0.05	50
Alg 9	0.10	5
Alg 10	0.10	10
Alg 11	0.10	20
Alg 12	0.10	50
Alg 13	0.20	5
Alg 14	0.20	10
Alg 15	0.20	20
Alg 16	0.20	50

$$\mathrm{RPD}(\%) = \frac{\mathrm{obj}^{\mathrm{avg}} - \mathrm{obj}^{\mathrm{best}}}{\mathrm{obj}^{\mathrm{best}}} \times 100 \tag{5}$$

where $\mathrm{obj}^{\mathrm{avg}}$ stands for the average objective function value of 5 runs of a specific version of the IG and $\mathrm{obj}^{\mathrm{best}}$ refers to the minimum sum of penalties among all algorithms. The results are then evaluated according to the average RPD values. For better visualization, Fig. 3 represents the mean plot of average RPD values with 95% confidence interval of different parameter levels.

In Fig. 3 it is possible to identify that smaller values of d' lead to better results since smaller values of RPD represent better solutions. Note that there is a clear difference between groups with distinct levels of d', e.g., algorithms 1, 2, 3, 4, which have an equivalent value of $d' = 0.01$ perform better in terms of solution quality than the other three groups consisting of algorithms 5, 6, 7, and 8 ($d' = 0.05$), algorithms 9, 10, 11, and 12 ($d' = 0.10$), and algorithms 13, 14, 15, and 16 ($d' = 0.20$). It is also worth mentioning that IG algorithms of these groups present confidence intervals of average RPD values that do not seem to overlap, which may be considered as a strong evidence that the group with $d' = 0.01$ has a more consistent performance in comparison with the others. For this reason, d' was chosen to be 0.01.

To determine $Temp$ it is fundamental to establish a range that guarantees the convergence of the IG algorithm. At a first moment, $Temp$ was set following the equation suggested by [27]:

$$Temp = T \times \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}}{n \times m \times 10} \tag{6}$$

Note that T is usually a real number in the interval [0, 1] that needs to be adjusted. The equation, however, did not seem to produce good results after a preliminary analysis. Taking instance $sch.20.4$ from the test set presented by [34] as an example, the sum of processing times is equal to 230. Now, supposing that $m = 2$ and $T = 1$, $Temp$ would have a value of approximately 0.003. After some empirical testing, it was observed that such a small value for $Temp$ yielded poor results. To deal with this issue, an empirical analysis was conducted for values of $Temp$ starting at 3000 and then decreased to obtain a reasonable range. This empirical analysis included different due window configurations, machine numbers as well as instance sizes. After some testing, it was identified that better results were achieved for values from 5 to 50.

Figure 3 shows the behavior of the different values of $Temp$. It is important to notice that algorithms with $Temp = 20$ presented a lower average RPD in comparison with the other levels; however, the confidence intervals for these averages overlap, which might indicate the proximity between the solution quality achieved by the algorithms. In addition, when analyzing the behavior of the confidence intervals, it is possible to observe a valley-shaped relation, that is, the average RPD starts to decrease from $Temp = 5$ to $Temp = 20$ when it starts to increase again. This fact corroborates with the notion that an ideal value for $Temp$ would lay between 10 and 50. For the sake of simplicity, the value of $Temp$ was set to 20 since it presented the lowest average RPD. Stated in other terms, the levels for $Temp$ and d' are set according to *Alg* 3 (see Table 2).

After calibrating the parameters d' and $Temp$, the IG algorithm was set to run with other CPU times as stopping criterion. Four different values for the parameter t are compared, and the CPU times are given as: $0.05 * n$, $0.10 * n$, $0.15 * n$, and $0.20 * n$. Figure 4 shows the mean plot for RPD values at 95% confidence intervals. It is possible to verify that the confidence intervals for the average RPD seem to overlap for all possible levels of t. This fact may indicate that the IG algorithm converges quickly. Note that the smallest RPD value is attained for $t = 0.20$, therefore, this valued is chosen in the experiments as the stop criterion.

5 Experimental Results

Since there is no heuristic reported for the problem under consideration, a numerical analysis is conducted between the 10 proposed algorithms. The experiments were conducted on a personal computer with an Intel® Core i5 processor running at 3.7GHz, 16GB of RAM, and implemented in C++ programming language. The benchmark instances are available

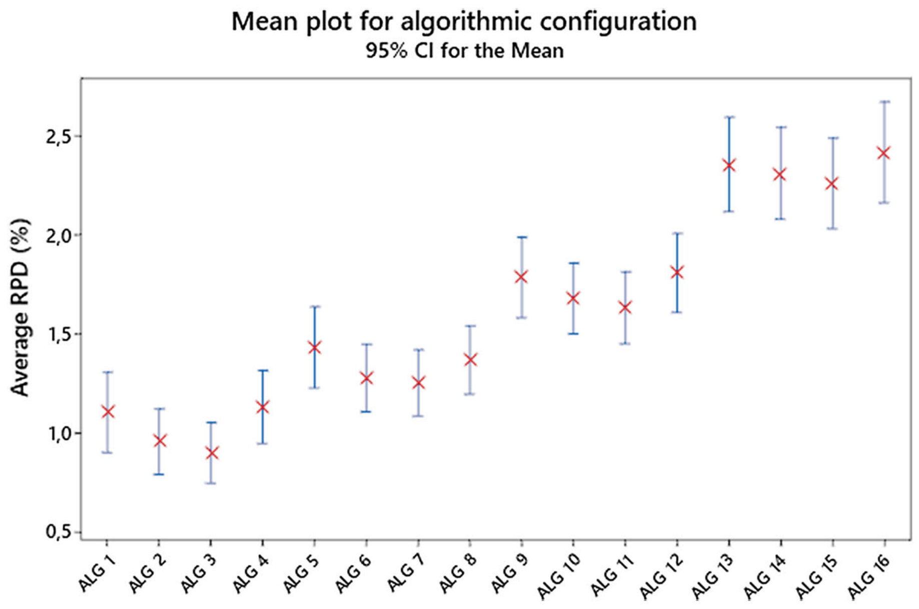


Fig. 3 Mean plot for algorithmic configuration of parameters $Temp$ and d'

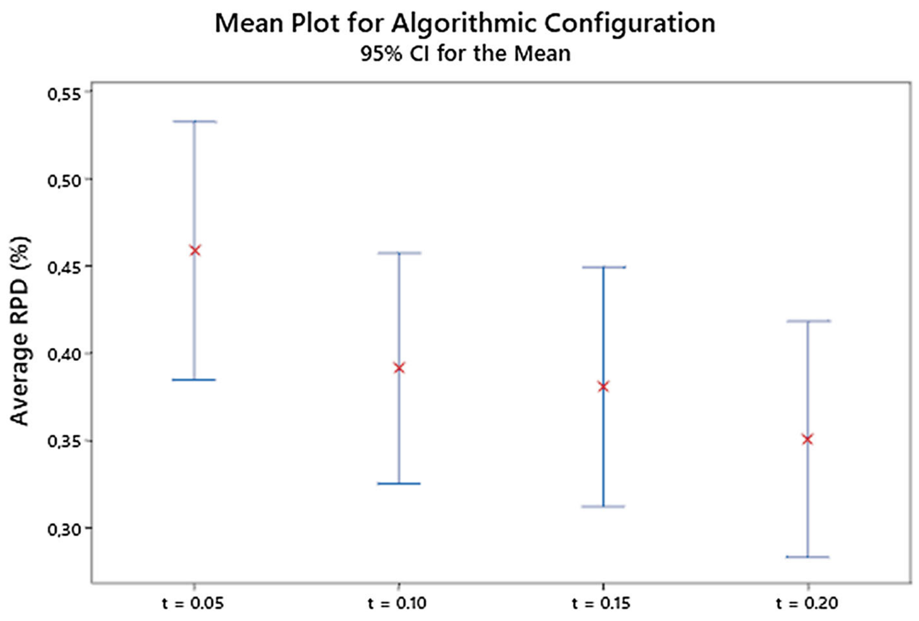


Fig. 4 Mean plot for algorithmic configuration of parameter t

at the well-known OR Library [1] [35]. The short Pascal generator as well as the problem instances were proposed for the analogous common due date single-machine problem [34]. The characteristics of the test set are seen in Table 3. We used 10 different instances from 5 sizes, 3 machine configurations, and 5 restriction factors combinations. Therefore, each algorithm solved a total of $10 \times 5 \times 3 \times 5 = 750$ instances. Two central performance measures were adopted, namely, the RPD and the computing times.

To demonstrate the performance of the heuristics, the average RPD values are compared based on three factors that modify the characteristics of the problem: instance size; window configuration; and number of machines. Table 4 summarizes the computational results of all proposed meth-

Table 3 Instance configuration

Factors	Levels
n	20, 50, 100, 200, 500
m	2, 4, 6
(h_l, h_r)	(0.1, 0.2), (0.1, 0.3), (0.2, 0.5), (0.3, 0.4), (0.3, 0.5)
α_j	$U[1, 10]$
β_j	$U[1, 15]$
p_j	$U[1, 20]$

[1] http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

Table 4 Average RPD values for each size of test instances

Algorithm	$n = 20$	$n = 50$	$n = 100$	$n = 200$	$n = 500$
RN-RGH-1	21.86	12.37	8.29	6.24	5.70
RN-RGH-2	21.08	14.69	11.87	10.08	10.75
RN-RGH-3	21.13	13.31	9.20	6.23	5.07
RN-RGH-4	17.45	12.09	8.12	6.01	5.06
RN-RGH-5	15.82	13.70	10.44	8.51	9.11
RN-SEA-1	3.87	2.98	2.10	1.51	1.76
RN-SEA-2	4.68	4.77	4.57	4.79	5.96
RN-SEA-3	4.74	4.42	3.56	1.85	1.08
RN-SEA-4	2.70	1.03	1.01	0.75	0.77
RN-SEA-5	5.69	2.58	1.65	2.00	3.05
$\mathrm{IG}_{\mathrm{SEA\text{-}4}}$	**1.03**	**0.17**	**0.10**	**0.12**	**0.24**

ods in terms of the solution quality for different instance sizes and the best results are marked in bold. There is a clear difference in the performance measures of the 11 heuristics, which stems from three major aspects: the initial priority rule used to order jobs before assigning them to machines; the sequencing strategy adopted on each machine; and the application of the stochastic local search (IG algorithm).

Clearly, the best performance is achieved for rule 4 (non-increasing β_j). When comparing the results for the same sequencing strategies, that is, between the 5 rules using the same sequencing algorithm (RN-RGH or RN-SEA), this rule outperforms the others for almost all instance sizes, except when combined with RN-RGH for solving instances with $n = 20$. Such an effect may be explained due to the nature of the data of the test set. Note that the individual tardiness penalty is allowed to be greater than the earliness penalty (refer to Table 3). In most practical contexts, contractual penalties incurred by a tardy job are usually higher than the cost of maintaining an early job in the inventory. Other possible explanation of the success of this rule relies on the fact that due windows are tight in relation to the sum of processing times. Following this rationale, a larger number of jobs should be scheduled in J_T; therefore, the amount of tardiness penalties is expected to be greater than the amount of earliness penalties. In terms of the improvements methods applied to sequence jobs in each machine, RN-SEA presented a consistently better performance. The reason for this behavior relates to the fact that RN-RGH might lead to poor results if there are many jobs with $\beta_j \gg \alpha_j$, where α_j is near to one and d_l is relatively small: in this situation, probably not all jobs with high tardiness penalties can be processed prior to the common due window and hence accumulate higher tardiness costs. Note that the good performance of RN-SEA also corroborates with the results previously reported in [26], when the application of a similar strategy to solve the analogous common due date single-machine problem presented better results in relation to several *ad hoc* heuristics and metaheuristics. Finally, the stochastic local search combined with the best performing strategy ($\mathrm{IG}_{\mathrm{SEA\text{-}4}}$) improved the average RPD for instances of all sizes. Table 4 also confirms that RN-SEA-4 is capable of exploring the search space in an efficient way. Note that, despite the fact that the IG algorithm finds better solutions in all cases, the RPD difference in relation to RN-SEA-4 is relatively small, which support the idea that it is possible to find good solutions with a constructive heuristic that is simple and do not require any parameter setting. A similar behavior is seen in relation to the number of machines as illustrated in Fig. 5.

It is worth noting that, as the number of machines increases, the common due window becomes more restricted. With this said, it is clear that RN-SEA-4 outperforms the other constructive algorithms. Note that the IG is more beneficial for a small number of machines. With a larger window, the stochastic local search is capable of finding better solutions since more jobs may be allocated to J_W. In this context, RN-SEA is proved to be more effective in relation to solution quality than RN-RGH.

Figure 6 also demonstrates the superiority of RN-SEA against RN-RGH for problems with different window configurations. Moreover, it is possible to observe that the good performance is maintained for all cases, regardless of the position where the common due window starts and its size. From Fig. 6, instances with more restrictive due windows (the ones that have a tighter left boundary factor h_l) show a smaller variation of RPD values and hence combining the IG with other heuristics may lead to smaller improvements. As the common due windows become looser, better gains are achieved, specially when their sizes are larger.

To understand the differences between the mean RPD values, it was conducted an analysis of variance (ANOVA). ANOVA allows to interpret the effects between groups that have been split on two or more independent variables on the dependent variable. In this case, the independent variables are the number of jobs n (A), the *Algorithm* (B), the *Window* configuration (C), and the number of machines m (D). The dependent variable is given as the solution quality, which is measured in terms of the RPD. The results of the ANOVA experiment are illustrated in Table 5. Note that the column source represents the independent variable and their interactions, i.e., row BD of the column source shows the interactions between *Algorithm* and m. The experiment was set to have a significance level $\alpha = 5\%$, and two hypotheses were tested:

- **H0**: the means of RPD are the same across the methods. In other words, there are no significant statistical differences between the means of the distributions of the RPD.
- **H1**: The means of the distributions of the RPD are different.

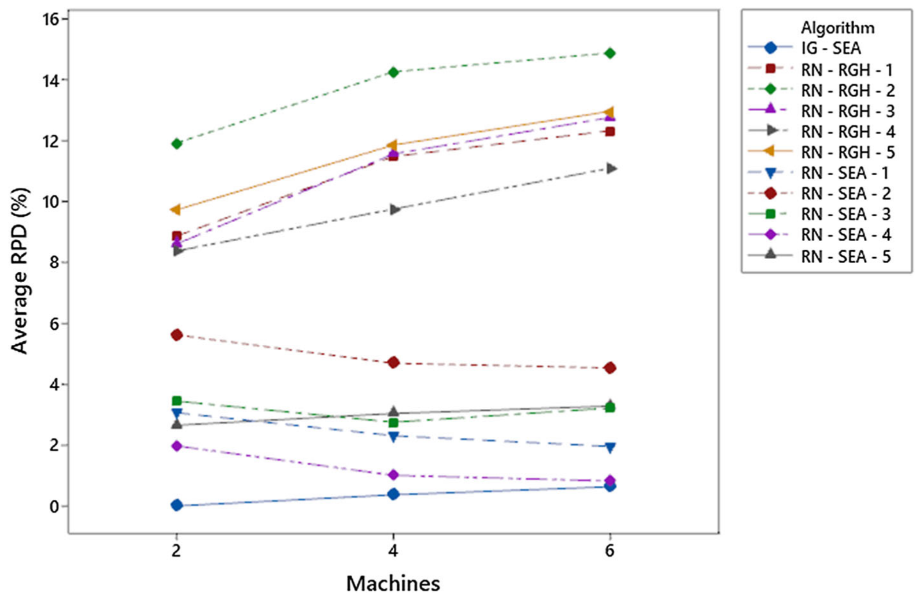


Fig. 5 Estimated marginal means of RPD values according to the number of machines

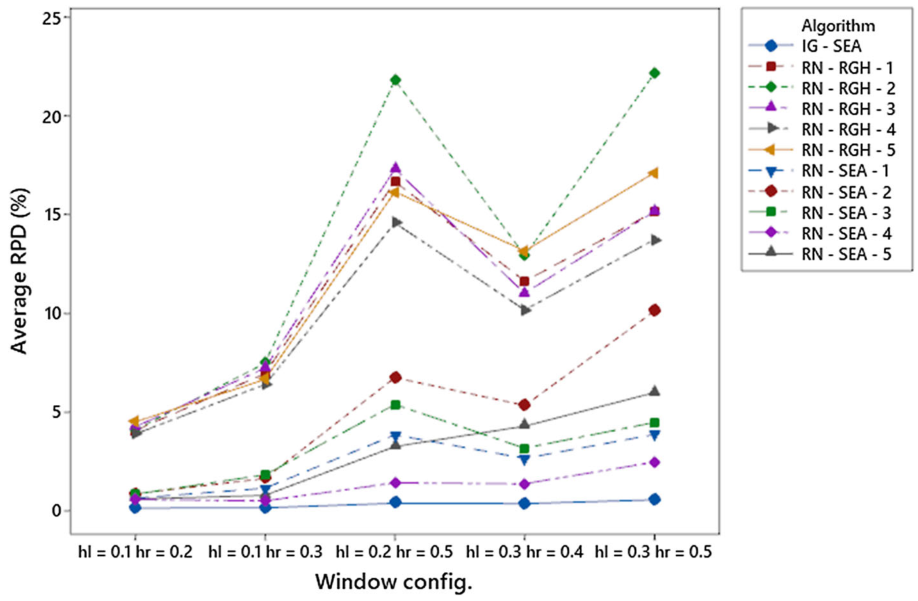


Fig. 6 Estimated marginal means of RPD values according to window configuration

From Table 5 it is possible to conclude that the interactions where the significance value is less than 0.05 indicate that at least one mean of the distributions of the RPD is different, that is, the null hypothesis can be rejected. It is worth noting that all of the interactions have different means, except ABCD. Consequently, this fact supports the idea that there are different means of RPD for the eleven algorithms. Moreover, it seems that the instance size affects the performance of the algorithms (row AB) together with the number of machines (row BD) and the window size (row BC).

Hitherto, it was identified the effects of different factors on the dependent variable RPD. The ANOVA experiment showed that there are statistically significant differences between means from various groups; however, it was not possible to detect in which elements from the groups these differences occur. This fact corroborates with the idea of applying a *post hoc* analysis. Here, this analysis is conducted by means of Tukey's honestly significant difference (HSD) test.

Table 6 shows the homogeneous subsets for the algorithms in relation to the dependent variable RPD for a sample size of 750, which relates to the best solution found by each algorithm for all considered instances. Note that two hypotheses being analyzed by Tukey's HSD are the same for the ANOVA experiment. Here, for the sake of simplicity, the algorithms are ordered in increasing order of the values of the means. For instance, $IG_{SEA\text{-}4}$ have a smaller average RPD value in comparison with RN-SEA-4, RN-SEA-1, and so on.

Table 5 Test of between-subjects effects

Source	Type III sum of squares	df	Mean square	F	Sig.
Corrected model	46.59	824	0.06	26.33	0.00
Intercept	35.29	1	35.29	16435.05	0.00
n (A)	4.99	4	1.25	580.41	0.00
Algorithm (B)	17.62	10	1.76	820.45	0.00
Window (C)	8.20	4	2.05	654.68	0.00
m (D)	0.24	2	0.12	54.85	0.00
AB	4.00	40	0.10	46.59	0.00
AC	1.96	16	0.12	56.95	0.00
AD	0.39	8	0.48	22.42	0.00
BC	4.05	40	0.10	47.12	0.00
BD	0.58	20	0.03	13.46	0.00
CD	0.14	8	0.17	7.98	0.00
ABC	2.43	160	0.02	7.07	0.00
ABD	0.83	80	0.10	4.84	0.00
ACD	0.21	32	0.01	3.01	0.00
BCD	0.37	80	0.01	2.12	0.00
ABCD	0.61	320	0.00	0.89	0.93
Error	15.94	7425	0.00		
Total	97.82	8250			
Corrected total	62.53	8249			

Table 6 Homogeneous subsets

	Subset for $\alpha = 5\%$						
Algorithm	1	2	3	4	5	6	7
$\text{IG}_{\text{SEA-4}}$	0.00						
RN-SEA-4	0.01	0.01					
RN-SEA-1		0.02	0.02				
RN-SEA-5			0.03				
RN-SEA-3			0.03				
RN-SEA-2				0.05			
RN-RGH-4					0.10		
RN-RGH-1					0.11	0.11	
RN-RGH-3						0.11	
RN-RGH-5						0.12	
RN-RGH-2							0.14
Sig.	0.36	0.07	0.78	1.00	0.09	0.86	1.00

With that said, it is evidenced that Tukey's HSD found 7 different homogeneous subsets, that is, groups of algorithms that share similar values of average RPD. The best performance is clearly achieved by $\text{IG}_{\text{SEA-4}}$; however, there is no statistically significant difference between the average RPD values found by RN-SEA-4. Observe that the last row of the table indicates the significance value for each subset. Taking subset 1 as an example, the significance value is 0.36, which is greater than 0.05 demonstrating that there is not enough evidence to reject the null hypothesis.

Table 6 also permits the interpretation that initial assignment rules similarly influence the performance of sequencing strategies. For instance, rule 4 is the best initial assignment strategy for both RN-SEA and RN-RGH. Moreover, two similar subsets are identified. Subsets 2 and 5 show that RN-SEA-4 and RN-SEA-1 together with RN-RGH-4 and RN-RGH-1 share comparable means. An analogous behavior occurs with subsets 3 an 6, which shows that, despite the fact that rules 1, 3, and 5 provide a relatively different performance when combined with RN-SEA and RN-RGH, there is not enough evidence indicating performance superiority of these rules in both sequencing strategies. Ultimately, it is possible to conclude that rule 2 has the worst performance of all initial assignment strategies as depicted in subsets 4 and 7.

Finally, other important measure refers to the computing efforts applied to obtain the solutions previously discussed. Here, we evaluate only the running times of RN-RGH and RN-SEA due to the fact that the IG algorithm uses the elapsed CPU time as a stop criterion.

When the heuristics are compared in terms of the sequencing strategy on each machine, it is observed that the family of RN-RGH heuristics requires less computational effort in relation to RN-SEA. This type of behavior occurs specially when the number of jobs exceeds 200, i.e., comparing two heuristics with the same initial assignment rules such as RN-

Table 7 Average CPU times in seconds (s)

Algorithm	n					m		
	20	50	100	200	500	2	4	6
RN-RGH-1	> 0.00	> 0.00	> 0.00	0.04	1.18	0.53	0.15	0.06
RN-RGH-2	> 0.00	> 0.00	> 0.00	0.06	1.91	0.87	0.24	0.09
RN-RGH-3	> 0.00	> 0.00	> 0.00	0.08	2.75	1.29	0.31	0.11
RN-RGH-4	> 0.00	> 0.00	> 0.00	0.04	0.75	0.33	0.10	0.05
RN-RGH-5	> 0.00	> 0.00	> 0.00	0.04	0.87	0.36	0.13	0.06
RN-SEA-1	> 0.00	> 0.00	0.03	0.73	37.57	20.63	1.90	0.47
RN-SEA-2	> 0.00	> 0.00	0.04	0.81	50.00	27.76	2.21	0.56
RN-SEA-3	> 0.00	> 0.00	0.05	1.10	79.96	45.31	2.76	0.60
RN-SEA-4	> 0.00	> 0.00	0.02	0.30	18.80	10.32	0.94	0.22
RN-SEA-5	> 0.00	> 0.00	0.01	0.09	3.41	1.72	0.28	0.11

RGH-3 and RN-SEA-3, it is seen that the average computing time of RN-RGH-3 for instances with 500 jobs is far inferior to the one in RN-SEA-3.

Moreover, as the number of machines increases, the computing times tend to decrease. Such a counter-intuitive result is also consistent with other studies in the same field [14,36]. The cause of this effect may relate to the smaller number of jobs assigned to each machine, which may reduce the number of combinations tested by the heuristics. In contrast with the instance size and the number of machines, the window configuration does not seem to alter the computing times. Table 7 summarizes the average CPU times in seconds according to instance size and number of machines.

6 Concluding Remarks

In this paper, we have developed a family of ten heuristics for scheduling parallel machines subject to common restrictive due windows. Moreover, we also showed an IG algorithm, which improved the existing solutions found by the best performing heuristic. The results were evaluated between the heuristics due to the fact that there is no other similar approach for the same problem.

As a matter of fact, only [14] proposed some exact solution procedures; however, they report that only instances with up to 40 jobs in size can be solved in reasonable time. They developed a set partitioning-type formulation to the problem under study. For its resolution, a column generation approach is addressed. An input required for this model is the generation of several single-machine sequences in each available machine. An exhaustive enumeration of such sequences could be prohibitive to solve large-sized or even medium-sized test instances. On the other hand, a small number of sequences could lead to low-quality solutions. The generation of sequences plays a key role in this method. In our view, this issue was not clearly explained or justified by [14]. Thereby, we decided not to consider this solution approach in our computational experiments.

This fact corroborates with the idea of proposing heuristics and metaheuristics to solve the problem under study, and, to the best of our knowledge, there were no similar approaches in the revised literature. Therefore, our objective was to develop easy implementation algorithms capable of solving instances with up to 500 jobs in size. Our computational studies demonstrate that the most time-consuming heuristic has an average CPU time of approximately 80 seconds, which confirms the possibility of application in practical contexts. Furthermore, analyzing the whole set of heuristics and their results allows us to notice some aspects in relation to the behavior of the algorithms:

– If the number of jobs is taken into consideration, it is possible to see clear differences in the performance of the heuristics. For all instance sizes, IG_{SEA-4} was the best performing method. In relation to the constructive heuristics, the best performance was achieved by RN-SEA-4. The reason for such effect may rely on the fact that the structure of the data favors the fourth assignment rule. Note that RN-RGH-4 is also the best performing heuristic regarding the number of jobs if it is taken into consideration heuristics with a similar sequencing principle on each machine, that is, the family of heuristics defined by RN-RGH. It is worth noting that the earliness and tardiness penalties are generated independently of each other and of the processing times. Thus, this produces all kinds of combinations in the characteristics of the jobs. Of course, changing the way in which the instances are generated may influence the performance of the algorithms.
– Looking at different phases of the algorithms, one can see that there is a clear difference in performance between RN-SEA and RN-RGH families. RN-SEA produces the best solutions, in many cases, independently of the initial ordering rule. In contrast, it also consumes more time to

find the solutions in relation to RN-RGH. Note that the initial assignment rules and the local search are the same for both families. This behavior occurs due to the fact that RN-RGH might lead to poor results if there are many jobs with $\beta_j \gg \alpha_j$, where α_j is near to one and d_l is relatively small.
- The IG algorithm has been designed as an alternative way for combining solutions in previous phases of the heuristics. One advantage of our method is related to the possibility of combining several algorithms to produce good-quality solutions while preserving reasonable computing times.

One line of future research would be to take advantage of designing parallel or cooperative algorithms or extending this study by adding different restrictions, such as jobs with distinct due windows, sequence-dependent setup times, and batching. It is also worth noting that some efforts could be directed to other production environments such as open shops or flow shops. Literature gaps are disclosed in the recent survey paper of [5], where an extensive review of scheduling problems involving common due dates and windows is conducted.

Author contributions GAR helped in investigation, software, formal analysis, writing—original draft, visualization; MSN contributed to conceptualization, writing—original draft, writing—review and editing, supervision, project administration, funding acquisition; BAP helped in writing—original draft, writing—review and editing, supervision, funding acquisition.

Funding This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This work was partially funded by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) through Grants 404232/2016-7, 306075/2017-2, 430137/2018-4, 130396/2018-4, and 303594/2018-7.

Data Availability and Material The test instances have been presented previously by Biskup and Feldmann [34]. All the evaluated test instances are available in this link: http://people.brunel.ac.uk/\protect\unhbox\voidb@x\penalty\@M\mastjjb/jeb/orlib/schinfo.html.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Baker, K.R.; Scudder, G.D.: Sequencing with earliness and tardiness penalties: a review. Oper. Res. **38**(1), 22–36 (1990). https://doi.org/10.1287/opre.38.1.22
2. Choi, B.C.; Park, M.J.: Single-machine scheduling with periodic due dates to minimize the total earliness and tardy penalty. J. Comb. Optim. **41**, 781–793 (2021)
3. Gordon, V.S.; Proth, J.M.; Chu, C.: Due date assignment and scheduling: SLK, TWK and other due date assignment models. Prod. Plann. Control **13**(2), 117–132 (2002). https://doi.org/10.1080/09537280110069621
4. Mor, B.; Mosheiov, G.: A note on the single machine CON and CONW problems with lot scheduling. J. Combinat. Optim. **42**, 327–38 (2021)
5. Rolim, G.A.; Nagano, M.S.: Structural properties and algorithms for earliness and tardiness scheduling against common due dates and windows: A review. Comput. Ind. Eng. (2020). https://doi.org/10.1016/j.cie.2020.106803
6. Biskup, D.; Feldmann, M.: On scheduling around large restrictive common due windows. Eur. J. Oper. Res. **162**(3), 740–761 (2005). https://doi.org/10.1016/j.ejor.2003.10.026
7. Janiak, A.; Janiak, W.A.; Krysiak, T.; Kwiatkowski, T.: A survey on scheduling problems with due windows. Eur. J. Oper. Res. **242**(2), 347–357 (2015). https://doi.org/10.1016/j.ejor.2014.09.043
8. Koulamas, C.: Single-machine scheduling with time windows and earliness/tardiness penalties. Eur. J. Oper. Res. **91**(1), 190–202 (1996). https://doi.org/10.1016/0377-2217(95)00116-6
9. Wu, Y.; Lai, K.K.: A production scheduling strategy with a common due window. Comput. Ind. Eng. **53**(2), 215–221 (2007). https://doi.org/10.1016/j.cie.2007.06.012
10. Mönch, L.; Unbehaun, R.; Choung, Y.I.: Minimizing earliness-tardiness on a single burn-in oven with a common due date and maximum allowable tardiness constraint. OR Spectrum **28**(2), 177–198 (2006). https://doi.org/10.1007/s00291-005-0013-4
11. Mönch, L.; Fowler, J.W.; Dauzère-Pérès, S.; Mason, S.J.; Rose, O.: A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. J. Sched. **14**(6), 583–599 (2011). https://doi.org/10.1007/s10951-010-0222-9
12. Rocholl, J.; Mönch, L.: Hybrid algorithms for the earliness-tardiness single-machine multiple orders per job scheduling problem with a common due date. RAIRO-Oper. Rese. **52**(4–5), 1329–1350 (2018). https://doi.org/10.1051/ro/2018029
13. Mula, J.; Bogataj, M.: OR in the industrial engineering of industry 4.0: experiences from the iberian peninsula mirrored in CJOR. Central Eur. J. Oper. Res. (2021). https://doi.org/10.1007/s10100-021-00740-x
14. Chen, Z.L.; Lee, C.Y.: Parallel machine scheduling with a common due window. Eur. J. Oper. Res. **136**(3), 512–527 (2002). https://doi.org/10.1016/s0377-2217(01)00068-6
15. Kramer, F.J.; Lee, C.Y.: Due window scheduling for parallel machines. Math. Comput. Modell. **20**(2), 69–89 (1994). https://doi.org/10.1016/0895-7177(94)90208-9
16. Mosheiov, G.; Oron, D.: Due-window assignment with unit processing-time jobs. Naval Res. Logist. **51**(7), 1005–1017 (2004). https://doi.org/10.1002/nav.20039
17. Mosheiov, G.; Sarig, A.: A note: a due-window assignment problem on parallel identical machines. J. Oper. Res. Soc. **62**(1), 238–241 (2011). https://doi.org/10.1057/jors.2009.179
18. Janiak, A.; Janiak, W.; Kovalyov, M.Y.; Werner, F.: Soft due window assignment and scheduling of unit-time jobs on parallel machines. 4OR Quart. J. Oper. Res. **10**(4), 347–360 (2012). https://doi.org/10.1007/s10288-012-0201-4
19. Gerstl, E.; Mosheiov, G.: An improved algorithm for due-window assignment on parallel identical machines with unit-time jobs. Inf. Process. Lett. **113**(19–21), 754–759 (2013b). https://doi.org/10.1016/j.ipl.2013.06.013
20. Mosheiov, G.; Sarig, A.: Scheduling identical jobs and due-window on uniform machines. Eur. J. Oper. Res. **201**(3), 712–719 (2010). https://doi.org/10.1016/j.ejor.2009.03.039
21. Gerstl, E.; Mosheiov, G.: Due-window assignment with identical jobs on parallel uniform machines. Eur. J. Oper. Res. **229**(1), 41–47 (2013a). https://doi.org/10.1016/j.ejor.2012.12.034

22. Janiak, A.; Janiak, W.; Kovalyov, M.Y.; Kozan, E.; Pesch, E.: Parallel machine scheduling and common due window assignment with job independent earliness and tardiness costs. Inf. Sci. **224**, 109–117 (2013). https://doi.org/10.1016/j.ins.2012.10.024
23. Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Kan, A.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discr. Math. **5**, 287–326 (1979)
24. Alvarez-Valdes, R.; Tamarit, J.M.; Villa, F.: Minimizing weighted earliness-tardiness on parallel machines using hybrid metaheuristics. Comput. Oper. Res. **54**, 1–11 (2015). https://doi.org/10.1016/j.cor.2014.08.020
25. Ying, K.C.; Lin, S.W.; Lu, C.C.: Effective dynamic dispatching rule and constructive heuristic for solving single-machine scheduling problems with a common due window. Int. J. Prod. Res. **55**(6), 1707–1719 (2017). https://doi.org/10.1080/00207543.2016.1224949
26. Lin, S.W.; Chou, S.Y.; Ying, K.C.: A sequential exchange approach for minimizing earliness-tardiness penalties of single-machine scheduling with a common due date. Eur. J. Oper. Res. **177**(2), 1294–1301 (2007). https://doi.org/10.1016/j.ejor.2005.11.015
27. Ruiz, R.; Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur. J. Oper. Res. **177**(3), 2033–2049 (2007). https://doi.org/10.1016/j.ejor.2005.12.009
28. Arroyo, J.E.C.; Leung, J.Y.T.: An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times. Comput. Ind. Eng. **105**, 84–100 (2017). https://doi.org/10.1016/j.cie.2016.12.038
29. Ribas, I.; Companys, R.; Tort-Martorell, X.: An iterated greedy algorithm for solving the total tardiness parallel blocking flow shop scheduling problem. Expert Syst. Applic. **121**, 347–361 (2019). https://doi.org/10.1016/j.eswa.2018.12.039
30. Lin, S.W.; Lu, C.C.; Ying, K.C.: Minimization of total tardiness on unrelated parallel machines with sequence- and machine-dependent setup times under due date constraints. Int. J. Adv. Manuf. Technol. **53**(1–4), 353–361 (2011b). https://doi.org/10.1007/s00170-010-2824-y
31. Ying, K.C.; Lin, S.W.; Huang, C.Y.: Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. Expert Syst. Applic. **36**(3), 7087–7092 (2009). https://doi.org/10.1016/j.eswa.2008.08.033
32. Kang, Q.; He, H.; Wei, J.: An effective iterated greedy algorithm for reliability-oriented task allocation in distributed computing systems. J. Parall. Distrib. Comput. **73**(8), 1106–1115 (2013). https://doi.org/10.1016/j.jpdc.2013.03.008
33. Lin, S.W.; Lee, Z.J.; Ying, K.C.; Lu, C.C.: Minimization of maximum lateness on parallel machines with sequence-dependent setup times and job release dates. Comput. Operat. Res. **38**(5), 809–815 (2011a). https://doi.org/10.1016/j.cor.2010.09.020
34. Biskup, D.; Feldmann, M.: Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. Comput. Oper. Res. **28**(8), 787–801 (2001). https://doi.org/10.1016/s0305-0548(00)00008-3
35. Beasley, J.E.: OR-library: Distributing test problems by electronic mail. J. Oper. Res. Soc. **41**(11), 1069–1072 (1990). https://doi.org/10.1057/jors.1990.166
36. Chen, Z.L.; Powell, W.B.: A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. Eur. J. Oper. Res. **116**(1), 220–232 (1999). https://doi.org/10.1016/s0377-2217(98)00136-2