Journal of the Operational Research Society

ISSN: 0160-5682 (Print) 1476-9360 (Online) Journal homepage: https://www.tandfonline.com/loi/tjor20

Exact and metaheuristic approaches for identical parallel machine scheduling with a common server and sequence-dependent setup times

João Marcos Pereira Silva, Ewerton Teixeira & Anand Subramanian

To cite this article: João Marcos Pereira Silva, Ewerton Teixeira & Anand Subramanian (2021) Exact and metaheuristic approaches for identical parallel machine scheduling with a common server and sequence-dependent setup times, Journal of the Operational Research Society, 72:2, 444-457, DOI: 10.1080/01605682.2019.1671153

To link to this article: https://doi.org/10.1080/01605682.2019.1671153

JOURNAL OF THE OPERATIONAL RESEARCH SOCIETY
2021, VOL. 72, NO. 2, 444–457
https://doi.org/10.1080/01605682.2019.1671153

THE OPERATIONAL RESEARCH SOCIETY
Taylor & Francis
Taylor & Francis Group

ORIGINAL ARTICLE

Exact and metaheuristic approaches for identical parallel machine scheduling with a common server and sequence-dependent setup times

João Marcos Pereira Silva[a], Ewerton Teixeira[a] and Anand Subramanian[b]

[a]Departamento de Engenharia de Produção, Centro de Tecnologia, Campus I, Cidade Universitária, Universidade Federal da Paraíba, João Pessoa, Brazil; [b]Departamento de Sistemas de Computação, Centro de Informática, Mangabeira, Universidade Federal da Paraíba, João Pessoa, Brazil

ABSTRACT

We consider the problem of scheduling parallel machines with a common server and sequence-dependent setup times, whose objective is to minimize the makespan. In this case, the common server, which can be a machine, individual or a team, is responsible for performing the setup operations. Therefore, there must be no conflicts while conducting them. An arc-time-indexed formulation is proposed for the problem, as well as an algorithm based on the metaheuristic iterated local search that makes use of an improved decoding algorithm. Computational experiments were carried out on 150 instances involving up to 100 jobs ranging from 2 to 10 machines. The results obtained suggest that the methods developed were capable of finding highly competitive solutions when compared to those achieved by existing approaches.

ARTICLE HISTORY
Received 17 January 2019
Accepted 10 September 2019

KEYWORDS
Scheduling; common server; arc-time-indexed formulation; iterated local search

1. Introduction

This paper deals with parallel machine scheduling with a common server and sequence-dependent setup times. Production environments with common server implies that at most one machine can be submitted to a setup operation at a time. When a setup is performed in a given machine, the other machines also requiring this type of operation must remain idle, thus potentially increasing the makespan value. As a result, this may affect the direct or indirect costs of processing the jobs.

Studies considering these characteristics are relevant because they are frequently identified in scenarios where the productivity is significantly affected by the setup times and also by the setup resources. This kind of application arises in companies that make use of flexible manufacturing systems, where for example, a robot that performs the setup operations are shared by the stations (machines), or even in printing industries, in which a setup team must work together to clean and reset a machine for processing the next job (Huang, Cai, & Zhang, 2010). The type of common server varies according to the environment and it can be a professional, a team or an equipment.

In this work, the objective is to minimize the makespan considering identical parallel machines and sequence-dependent setup times. According to the classification scheme by Hall, Potts, and Sriskandarajah (2000), the existence of a server is specified in the first (α) field. Therefore, in the three-field notation proposed by Graham, Lawler, Lenstra, and Kan (1979), the problem is denoted as $P, S_1|s_{ij}|C_{max}$. Such problem is clearly $\mathcal{NP}$-hard as it includes the well-known traveling salesman problem as special case when only a single machine is considered and the processing times of the jobs are disregarded.

Most of the related works that consider a common server deal with the case where only two identical machines are available, and they also assume that the setups are not sequence-dependent. Abdekhodaee, Wirth, and Gan (2004) proposed heuristic algorithms for two special cases of problem $P_2, S_1|s_j|C_{max}$, where jobs have identical processing times and the setup times are also equal. Abdekhodaee, Wirth, and Gan (2006) developed a greedy heuristic and a genetic algorithm (GA) for problem $P_2, S_1|s_j|C_{max}$. Zhang and Wirth (2009) applied a list scheduling algorithm for three special cases of the online version of problem $P_2, S_1|s_j|C_{max}$. Gan, Wirth, and Abdekhodaee (2012) presented two mixed integer linear programming (MILP) formulations and two branch-and-price (B&P) approaches for problem $P_2, S_1|s_j|C_{max}$. Hasani, Kravchenko, and Werner (2014a) studied problem $P_2, S_1|s_j|C_{max}$, where they proposed two MILP models based on the idea of decomposing the scheduling of jobs in a set of blocks. Later, Hasani, Kravchenko, and Werner (2014b) proposed a GA and a simulated

CONTACT João Marcos Pereira Silva joaomarkco@gmail.com Departamento de Engenharia de Produção, Centro de Tecnologia, Campus I, Cidade Universitária, Universidade Federal da Paraíba, João Pessoa-PB, CEP, 58059-900, Brazil

annealing (SA) capable of solving up to 1000-job instances of the same problem.

In contrast, few works dealt with an arbitrary number of machines and/or sequence-dependent setup times. Problem $PD, S_1|s_{ij}|C_{max}$ was studied by Huang et al. (2010), where they proposed a mathematical model and a hybrid GA. Kim and Lee (2012) developed two MILP formulations and a hybrid algorithm that combines tabu search (TS) and SA to solve problem $P, S_1|s_j|C_{max}$. Hamzadayi and Yildiz (2016) put forward a SA and a dispatching rules-based complete rescheduling approach for the dynamic version of problem $P, S_1|pmtn, s_{ij}|C_{max}$. A concise review on scheduling problems with a common server can be found in Hamzadayi and Yildiz (2017).

To our knowledge, the work by Hamzadayi and Yildiz (2017) was the only one to tackle the same variant considered in our study. They proposed a MILP model, as well as two heuristics based on SA and GA. Both algorithms rely on a decoding procedure that builds a feasible solution from a sequence of jobs. Computational experiments were carried out on instances with up to 100 jobs.

The main contributions of this work are as follows.

- We propose a new integer programming model, which consists of an arc-time-indexed formulation, for problem $P, S_1|s_{ij}|C_{max}$, Computational experiments revealed that the developed formulation was capable of outperforming the model by Hamzadayi and Yildiz (2017) both in terms of number of optimal solutions found and quality of the lower bounds. While their formulation could only solve instances with up to 9 jobs, ours managed to prove the optimality of instances involving up to 21 jobs.
- We present a simple yet effective iterated local search (ILS) (Lourenço, Martin, & Stützle, 2019) algorithm for the problem with a view of obtaining high quality solutions in an acceptable CPU time. We believe this is the first time ILS is employed to solve a scheduling problem involving a common server. When comparing the results obtained by our metaheuristic with the GA and SA algorithms by Hamzadayi and Yildiz (2017), one can observe that the proposed methodology is capable of producing better results both in terms of solution quality and CPU time.
- A novel and more efficient implementation of the decoding procedure originally developed in Hamzadayi and Yildiz (2017) is introduced. We show by means of computational experiments that the enhanced implementation yields a significant decrease in the CPU time required by the proposed metaheuristic, when compared to the original implementation.

The remainder of the paper is organized as follows. Section 2 formally describes the problem considered in this study. Section 3 introduces the arc-time-indexed formulation. Section 4 explains the proposed metaheuristic approach as well as the improved decoding algorithm. Section 5 reports and discusses the computational results. Finally, Section 6 contains the concluding remarks.

2. Problem description

Let $J = \{1, ..., n\}$ be the set of jobs to be scheduled in a set $M = \{1, ..., m\}$ of identical parallel machines. Each job $j \in J$ has an associated processing time p_j and s_{ij} is the setup time incurred if $j \in J$ is scheduled immediately after $i \in J$ in the same machine. A common server is in charge or performing the setup operations and they cannot be carried out simultaneously in more than one machine. Therefore, idle times may occur so as to generate feasible solutions. It is assumed that all jobs are available at time instant 0, and that is not necessary to perform a setup operation before processing the first job scheduled in a machine. Each job must be scheduled exactly once and preemption is not allowed. The objective is to minimize the makespan.

An example of a feasible schedule of an instance involving 12 jobs and 3 machines is depicted in Figure 1. The data of the instance is given in Figure 1. By observing the solution one can verify that the makespan value is 296. Note that in this case machines 2 and 3 are expected to be idle after processing jobs 2 and 8, respectively. The information provided in both Table 1 and Figure 1 will be reused further in the next sections.

3. Arc-time-indexed formulation

Arc-time-indexed formulations revealed to be an interesting alternative for solving a variety of scheduling problems (Nesello, Subramanian, Battarra, & Laporte, 2018; Pessoa, Uchoa, Poggi de Aragão, & Rodrigues, 2010; Silva, Subramanian, & Pessoa, 2018). In what follows, we explain how the problem can be modeled using this type of approach.

The proposed formulation assumes a time horizon from 0 to T, where T is an upper bound on the maximum completion time of a job in an optimal solution. Each variable is associated with an arc of a network where the nodes have job and time indices.

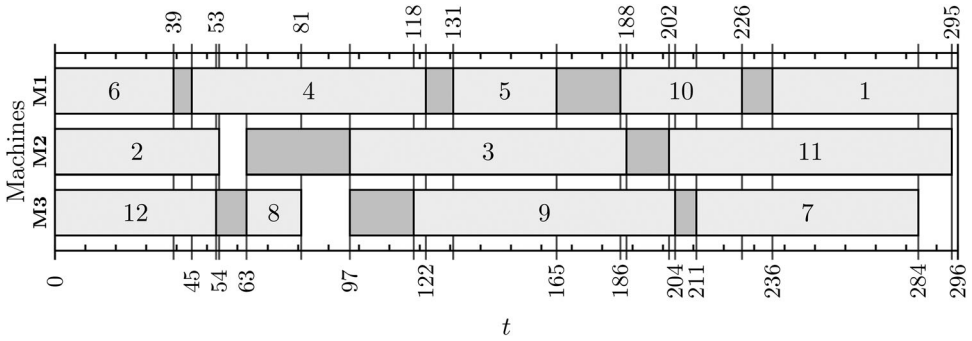


Figure 1. A feasible schedule of 12 jobs on 3 machines.

Table 1. Processing times and sequence-dependent setup times for a instance with 12 jobs and 3 machines.

Jobs	Processing Time	Sequence–dependent setup times											
		1	2	3	4	5	6	7	8	9	10	11	12
1	60	–	30	34	19	28	20	15	18	27	13	26	27
2	54	13	–	34	23	21	18	22	18	15	11	23	31
3	91	22	25	–	16	12	16	13	11	6	25	14	9
4	77	22	21	25	–	9	21	14	12	14	14	17	26
5	34	13	27	23	7	–	15	12	10	5	21	8	21
6	39	24	19	10	6	11	–	20	16	16	20	9	19
7	73	27	27	30	33	26	30	–	12	31	34	15	12
8	18	28	15	18	31	24	30	28	–	21	23	29	27
9	86	19	20	19	13	6	10	7	5	–	27	8	19
10	40	10	17	17	12	18	7	21	12	14	–	13	20
11	93	29	22	17	24	17	18	28	7	22	19	–	26
12	53	27	25	20	34	28	28	11	10	22	33	26	–

This network is defined over an acyclic graph $G = (V = N \cup D, A = A^1 \cup A^2 \cup A^3 \cup A^4)$, where set $N = \{(j,t) : j \in J, t = 0, ..., T - p_j\}$ contains the vertices associated with the actual jobs and $D = \{(0,t) : t = 0, ..., T\}$ is the set of vertices associated with the dummy job 0. For modeling purposes, we assume w.l.o.g. that idle times occur before the job is processed. We also assume that job j can start to be processed when arriving at vertex (j, t), but not necessarily at instant t due to the possible existence of idle times. The dummy job 0 is the first and last job to be processed in each machine, with $p_0 = 0$ and $s_{0j} = s_{j0} = 0, \forall j \in J$. For convenience, the arc $((i, t - p_i - s_{ij})(j,t))$ is denoted as (i, j, t). The schedule of each machine is represented by a path that starts at node (0, 0) and ends at node $(0,t), t \leq T$. The sets A^1, A^2, A^3 and A^4 are defined as follows:

- $A^1 = \{(i,j,t) = ((i, t - p_i - s_{ij}) \quad (j,t)) : (i, t - p_i - s_{ij}) \in N, (j,t) \in N, j \in J \setminus \{i\}\}$ contains the arcs connecting the vertices from N between themselves;
- $A^2 = \{(0,j,0) = ((0,0)(j,0)) : (j,0) \in N\}$ contains all arcs connecting the vertices from D to N;
- $A^3 = \{(j,0,t) = ((j, t - p_j - s_{j0})(0,t)) : (j, t - p_j - s_{j0}) \in N, (0,t) \in D, j \in J\}$ contains all arcs connecting the vertices from N to D;
- $A^4 = \{(j,j,t) = ((j,t-1)(j,t)) : (j,t-1) \in N, (j,t) \in N\}$ contains the arcs associated with idle times. Arcs (j, j, t) store the information related to job j, while it remains idle, thus allowing the setup time from j to its immediate successor to be computed correctly (Silva et al., 2018).

Let $c_a = c(i,j,t)$ be the cost of an arc $a = (i,j,t) \in A^1 \cup A^2$, if job j starts to be processed at instant $t : c(i,j,t) = f_j(t + p_j)$. Note that $c_a = 0, \forall a \in A^3 \cup A^4$. Set $N^j = \{(i,t) \in N : i = j\}$ contains all arcs associated with job j. For each vertex $v \subset V, \delta^-(v)$ and $\delta^+(v)$ denote the arcs entering and leaving v, respectively. The proposed arc-time-indexed formulation (ATIF) is as follows.

$$\min \quad C_{max} \tag{1}$$

Subject to

$$\sum_{a \in \delta^-(N^j) \setminus A^4} x_a = 1, \forall j \in J \tag{2}$$

$$\sum_{a \in A^2} x_a = m, \tag{3}$$

$$\sum_{a \in \delta^-(\{v\})} x_a - \sum_{a \in \delta^+(\{v\})} x_a = 0, \forall v \in V \setminus \{(0,0),(0,t)\} \tag{4}$$

$$C_{max} \geq \sum_{a \in \delta^-(N^j)} c_a x_a, \forall j \in J \tag{5}$$

$$\sum_{a' \in D_t} x_{a'} \leq 1, t = 0, ..., T-1 \tag{6}$$

$$x_a \in \{0,1\}, \forall a \in A \tag{7}$$

Objective function (1) minimizes the *makespan*. Constraints (2) indicate that each job must be

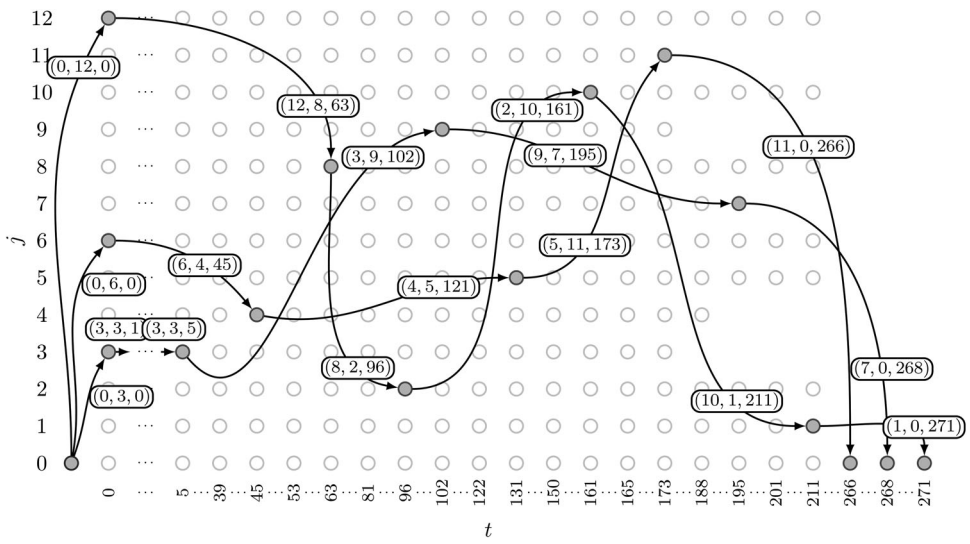


Figure 2. Path representation of the optimal solution obtained by ATIF for the example instance in the layered network.

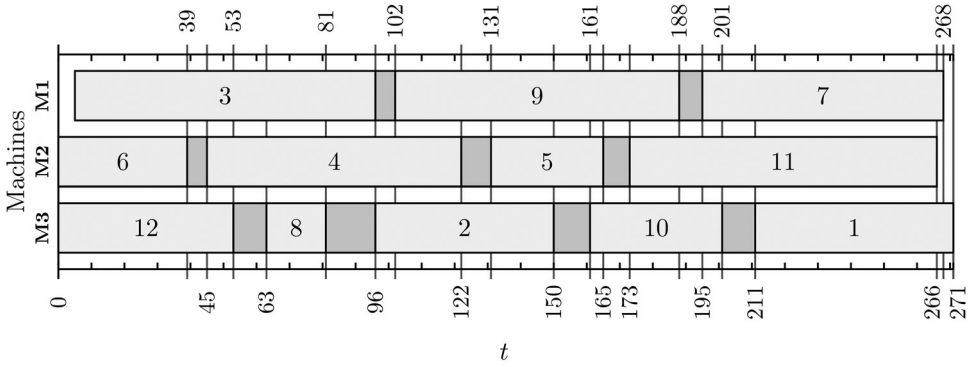


Figure 3. Optimal schedule obtained by ATIF for the example instance.

processed exactly once. Constraint (3) guarantees that m machines will be used for processing the jobs. Constraints (4) ensure the flow conservation. Constraints (5) compute the makespan. Constraints (6) prevent the occurrence of more than one setup operation at a time. Arc a' is defined by (i, j, t') and D_t is the set of variables $x_{a'}$ that when active will imply in the occurrence of a setup at time t. Constraints (7) define the domain of the variables.

The network representation and Gantt chart associated with the optimal solution of the instance introduced in Section 2 is illustrated in Figures 2 and 3, respectively.

In this case, the makespan value is 271 and the solution is composed by non-zero variables x_a associated with the following arcs: (0, 3, 0), (1–5), (3, 9, 102), (9, 7, 195), (7, 0, 268), (0, 6, 0), (6, 4, 45), (4, 5, 121), (5, 11, 173), (11, 0, 266), (0, 12, 0), (12, 8, 63), (8, 2, 96), (2, 10, 161), (10, 1, 211), (1, 0, 271). The non-zero variables associated with arcs (1–5) indicate that machine 1 remains idle between times 1 and 5, thus postponing the processing starting time of job 3. This happened in order to avoid conflict between the setup operations of job 2 (machine 3) and job 9 (machine 1). For convenience, the graph only contains the necessary nodes for representing the solution. Also, the nodes and arcs associated with (2–4) were suppressed.

4. Metaheuristic approach

This section describes the proposed ILS-based algorithm, including the neighborhood and perturbation operators, as well as the modified decoding procedure.

4.1. Solution representation

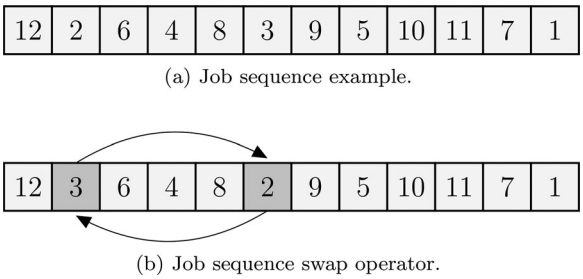
A solution is represented by a job priority sequence $(J_1, J_2, J_3, \ldots, J_{|J|})$ whose cost is computed by applying a decoding algorithm, where the jobs are iteratively assigned to the machine that yields the smallest completion time. Our decoding algorithm is based on the one by Hamzadayi and Yildiz (2017). For brevity, we denote the original and modified decoding algorithms as ODA (Algorithm 1) and MDA (Algorithm 2), respectively.

Algorithm 1. Original Decoding Algorithm

1: **procedure** ORIDECALGORITHM($s(.)$)
2: $S_{j,m} \leftarrow 0\ \forall j \in J, \forall m \in M$ ▷ Time at which machine m starts to process job j
3: $E_{j,m} \leftarrow 0\ \forall j \in J, \forall m \in M$ ▷ Time at which machine m completes the processing of job j
4: $CS \leftarrow 0$ ▷ Time at which the common server becomes available
5: **for** $k \leftarrow 1$ to $|J|$ **do**
6: $j \leftarrow s(k)$ ▷ Job at the k-th position of the job priority sequence
7: $f' \leftarrow \infty$
8: **for all** $m \in M$ **do**
9: **if** $E_{j,m} = 0$ **then**
10: **if** $f' \geq p_j$ **then**
11: $f' \leftarrow p_j$
12: $m' \leftarrow m$
13: **else**
14: **if** $CS \leq E_{j,m}$ **then**
15: **if** $f' \geq E_{j,m} + s_{ij} + p_j$ **then** ▷ i is the job that was last assigned to machine m
16: $f' \leftarrow E_{j,m} + s_{ij} + p_j$
17: $m' \leftarrow m$
18: **else**
19: **if** $f' \geq CS + s_{ij} + p_j$ **then**
20: $f' \leftarrow CS + s_{ij} + p_j$
21: $m' \leftarrow m$
22: **if** $E_{j,m'} = 0$ **then**
23: $S_{j,m'} \leftarrow 0$
24: $E_{j,m'} \leftarrow p_j$
25: **else**
26: **if** $CS \leq E_{j,m'}$ **then**
27: $S_{j,m'} \leftarrow E_{j,m'} + s_{ij}$
28: **else**
29: $S_{j,m'} \leftarrow CS + s_{ij}$
30: $E_{j,m'} \leftarrow S_{j,m'} + p_j$
31: $CS \leftarrow S_{j,m'}$
32: **return** $\max\{E_{j,m'} | j \in J, m \in M\}$

Algorithm 2. Modified Decoding Algorithm

1: **procedure** MODDECALGORITHM($s(.), f(s')$)
2: $A_j \leftarrow \emptyset\ \forall j \in J$ ▷ Machine where job j must be processed
3: $S_j \leftarrow 0\ \forall j \in J$ ▷ Time at which the job j starts to be processed
4: $E_m \leftarrow 0\ \forall m \in M$ ▷ Completion time of machine m
5: $CS \leftarrow 0$
6: **for** $k \leftarrow 1$ to $|J|$ **do**
7: $j \leftarrow s(k)$
8: $f' \leftarrow \infty$
9: **for all** $m \in M$ **do** ▷ If the machine m has no job assigned, then $s_{ij} = 0$
10: **if** $f' > \max(E_m, CS) + s_{ij} + p_j$ **then**
11: $f' \leftarrow \max(E_m, CS) + s_{ij} + p_j$
12: $m' \leftarrow m$
13: **if** $f' > f(s')$ **then**
14: $E_{m'} \leftarrow f'$
15: **go to** line 20
16: $A_j \leftarrow m'$
17: $S_j \leftarrow \max(E_{m'}, CS) + s_{ij}$
18: $E_{m'} \leftarrow f'$
19: $CS \leftarrow S_j$
20: **return** $\max\{E_m | m \in M\}$



(a) Job sequence example.

(b) Job sequence swap operator.

Figure 4. Job sequence and swap operator examples.

The algorithms work as follows: (i) the Gantt chart structure is initialized (ODA: lines 2–4; MDA: lines 2–5); (ii) the machine that yields the smallest completion time after inserting job j is identified (ODA: lines 8–21; MDA: lines 9–12); (ii) the job j is assigned to the end of the sequence associated with the machine selected in the previous step (MDA: line 16); and (iv) the Gantt chart structure is updated (ODA: lines 22–31; MDA: lines 17–19). Once all jobs are assigned (ODA: lines 5–31; MDA: lines 6–19) the value of the makespan is returned (ODA: line 32; MDA: line 20).

Note that in addition to the job priority sequence, MDA receives the value of the makespan $f(s')$ associated with the best incumbent solution s' of the current local search iteration (see Subsection 4.2) as an input value. This allows for interrupting the MDA execution if the cost of the solution under evaluation becomes greater than $f(s')$ (MDA: lines 13–15), which may reduce the total CPU time by avoiding unnecessary computations. Both algorithms have complexity $\mathcal{O}(nm)$ but MDA runs faster in practice because, in addition to the interruption strategy, it clearly performs less operations and selection statements.

It is worth mentioning that the pseudocode of the ODA presented in Hamzadayi and Yildiz (2017) contains some issues. In lines 14 and 26 of Algorithm 1, the authors actually wrote "$CS = 0$", which would allow the overlapping of jobs in the same machine. This is because CS stores the last time instant that a setup operation has been performed, and the completion time of the last job assigned to the same machine could be greater than CS. For example, the schedule depicted in Figure 1 is generated using the job priority sequence presented in Figure 4(a). Note that job 8 is the fifth in the sequence and before its assignment we have $CS = 45$. The cost would then be computed with the setup of job 8 starting at this time instant, which will result in an overlap with job 12, thus yielding an infeasible solution. Therefore, it is necessary to replace "$CS = 0$" with "$CS \leq E_{j,m}$"so as to prevent this from happening.

4.2. Proposed algorithm

The proposed algorithm, called ILS_{CS}, is a multi-start procedure based on ILS. We decided to implement this metaheuristic not only because we could not identify any other ILS-based approach for problems involving a common server, but also due its successful performance in solving other scheduling problems with sequence-dependent setup times (Kramer & Subramanian, 2017; Silva et al., 2018; Subramanian, Battarra, & Potts, 2014; Subramanian & Farias, 2017). The flowchart of ILS_{CS} is presented in Figure 5. The algorithm receives as input parameters I_R and I_{ILS}, which respectively refer to the maximum number of restarts and consecutive perturbations without improvements of the ILS procedure.

The multi-start heuristic starts by generating an initial solution using a completely randomized insertion procedure. Next, a local search is performed using the swap operator, as described in Algorithm 3. Note that jobs from the first m positions are only swapped with those after the m-th position so as to avoid generating symmetric solutions when applying the decoding algorithm, which in turn, assigns the first m jobs of the sequence to each machine. We chose to adopt the first improvement strategy because it helped speeding up the local search procedure without compromising the solution quality when compared to the best improvement strategy. The overall complexity of enumerating ($\mathcal{O}(n^2)$) and evaluating ($\mathcal{O}(nm)$) all swap moves is of the order of $\mathcal{O}(n^3m)$. Moreover, whenever a solution is modified, either due to a local search move or a perturbation move, the search restarts. It is worth mentioning that other neighborhood structures such as job reinsertion were implemented but preliminary experiments revealed that they did not contribute to improve the solution quality. Therefore, we chose to use only the swap neighborhood. An example of the application of a swap move can be found in Figure 4, where jobs 3 and 2 are interchanged. It can be observed that by applying the decoding algorithm over the resulting sequence depicted in Figure 4(b), one obtains the optimal solution presented in Figure 3.

Furthermore, if a local optimal solution is found and the maximum number of consecutive perturbations without improvements is smaller than I_{ILS} (i.e., the ILS stopping criterion is not met), then the algorithm modifies the solution of the current multi-start iteration by means of a perturbation mechanism and the search continues from this perturbed solution. The perturbation mechanism consists of multiple swaps. More specifically, 1, 2 or 3 pairs of distinct jobs are randomly chosen and then swapped. If the ILS

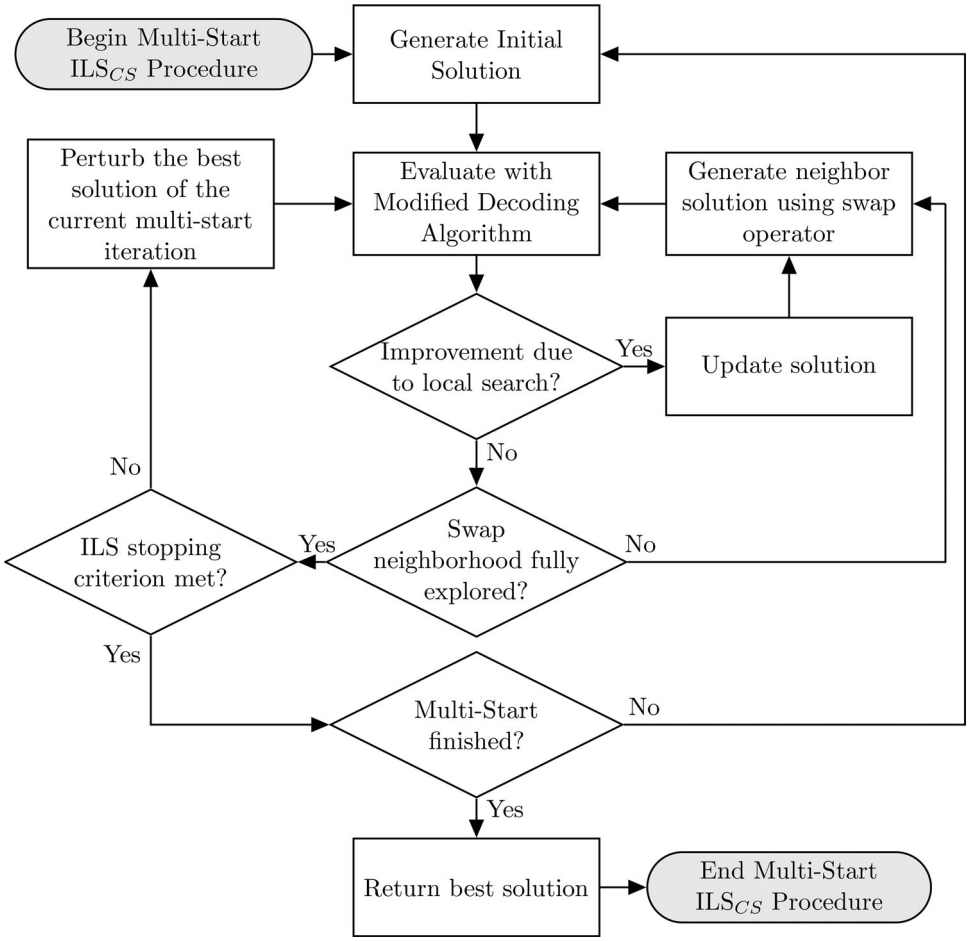


Figure 5. Flowchart of the proposed algorithm.

stopping criterion is met, then the algorithm restarts from the beginning. If the maximum number of restarts is achieved then ILS$_{CS}$ terminates and it returns the best solution found.

One important observation is that we do not allow MDA to be interrupted when computing the cost of an initial or perturbed solution because in these cases we must determine the cost anyway. Hence, in such situations we simply provide a sufficiently large number as an input cost for the procedure. This will naturally prevent MDA to be prematurely interrupted.

Algorithm 3. Local Search

```
1: procedure LocalSearch(s)
2:     for i ← 1 to |J|−1 do
3:         k ← max(|M|, i) + 1
4:         for j ← k to |J| do
5:             s′ ← Swap(s(i), s(j))          ▷ Jobs at i,j-th positions are swapped
6:             f ← ModDecAlgorithm(s′, f(s))
7:             if f < f(s) then
8:                 s ← s′
9:                 go to line 2
10: return s
```

5. Computational experiments

The exact and heuristic approaches were implemented using C++ (g++ 5.4.0) and all computational experiments were executed in single thread of an Intel® Core™ i7-3770 with 3.40 GHz and 16 GB of RAM running Ubuntu Linux 16.04 operating system. CPLEX 12.7 was used as a MILP solver.

Because the benchmark instances suggested in Hamzadayi and Yildiz (2017) were not made available by the authors, we had not only to generate new ones but also to reimplement their MILP formulation as well as the proposed SA and GA algorithms, with a view of properly evaluating the performance of our methods. The metaheuristics were executed 10 times in all experiments reported in this section.

Table 2. Aggregate average results found by different exact methods.

g	m	n	Gap (%)								CPU (s)	
			ATIF				MILP				ATIF	MILP
			#Opt	Min	Avg	Max	#Opt	Min	Avg	Max		
1	2	6	5	0.00	0.00	0.00	5	0.00	0.00	0.00	0.1	0.9
2		8	5	0.00	0.00	0.00	5	0.00	0.00	0.00	0.6	157.6
3		10	5	0.00	0.00	0.00	0	46.67	51.20	54.13	3.4	3600
4		14	5	0.00	0.00	0.00	0	75.00	76.22	77.69	198.7	3600
5		20	0	46.68	46.95	47.49	0	82.24	83.99	85.56	3600	3600
6	3	9	5	0.00	0.00	0.00	5	0.00	0.00	0.00	0.3	397.2
7		12	5	0.00	0.00	0.00	0	52.40	56.54	61.33	4.3	3600
8		15	5	0.00	0.00	0.00	0	66.32	69.29	73.24	45.1	3600
9		21	1	0.00	26.28	33.17	0	75.97	78.94	80.86	3488.1	3600
10		30	0	47.59	47.89	48.42	0	82.85	83.60	84.78	3600	3600
11	4	12	5	0.00	0.00	0.00	0	43.66	46.93	53.71	1.1	3600
12		16	5	0.00	0.00	0.00	0	58.26	62.63	65.79	76.5	3600
13		20	4	0.00	5.31	26.54	0	70.30	71.55	73.39	1989.9	3600
14		28	0	47.06	47.59	47.88	0	76.26	76.81	77.29	3600	3600
15		40	0	–	–	–	0	82.36	84.36	85.33	3600	3600
16	5	15	5	0.00	0.00	0.00	0	42.60	46.71	49.73	7.0	3600
17		20	5	0.00	0.00	0.00	0	54.88	60.78	65.64	626.1	3600
18		25	0	27.20	30.89	32.28	0	58.95	68.41	72.27	3600	3600
19		35	0	47.82	48.12	48.43	0	75.49	76.96	79.60	3600	3600
20		50	0	–	–	–	0	83.50	84.64	85.21	3600	3600
21	7	21	5	0.00	0.00	0.00	0	39.24	45.82	54.85	420.0	3600
22		28	0	22.82	26.81	29.82	0	55.81	61.32	68.62	3600	3600
23		35	0	33.40	41.50	47.91	0	64.00	68.82	71.79	3600	3600
24		49	0	48.64	48.64	–	0	76.91	78.06	79.63	3600	3600
25		70	0	–	–	–	0	–	–	–	3600	3600
26	10	30	0	14.84	17.58	20.56	0	43.86	46.58	49.20	3600	3600
27		40	0	31.72	32.13	32.78	0	58.26	61.14	63.50	3600	3600
28		50	0	48.40	48.85	49.38	0	–	–	–	3600	3600
29		70	0	–	–	–	0	–	–	–	3600	3600
30		100	0	–	–	–	0	–	–	–	3600	3600

5.1. Benchmark instances

The benchmark instances were generated as in Balakrishnan, Kanet, and Sridharan (1999); Hamzadayi and Yildiz (2017). The number of machines varied from 2 to 10, namely, $m = 2$, 3, 4, 5, 7 and 10, whereas the number of jobs was chosen as a function of the number of machines, namely, $n = 3m$, $4\,m$, $5\,m$, $7\,m$ and $10\,m$, thus leading to a total of 30 groups of instances. Five test-problems were generated for each group, resulting in a total of 150 instances. The processing and setup times were uniformly generated from the interval $[10, 100]$ and $[5, 50]$, respectively. The values of the setup times were further modified so as to respect the triangle inequality: $s_{i,j} \leq s_{i,k} + s_{k,j}, \forall i \neq j \neq k \in J$.

5.2. Parameter tuning

In order to calibrate the main parameters of ILS_{CS}, i.e., I_R and I_{ILS}, we have selected a set of 15 challenging instances. At first, we conducted tuning experiments to choose a value for I_{ILS}. We tried different values for this parameter, namely, 50, 100 and 150, while keeping $I_R = 1$. The average gaps were computed between the average solution and the best solution found during this experiment. We also considered the average CPU time. From the results obtained, we decided to adopt $I_{ILS} = 100$ as it offered a reasonable trade-off between solution quality and CPU time.

The next step was to conduct tuning experiments to choose a value for I_R. We tried three different values: 5, 10 and 15. We decided to select $I_R = 10$ because we believe that it provides a good compromise between solution quality and CPU time.

5.3. Results

We start by reporting the average aggregate results obtained by the MILP formulations on each group of instances and afterwards we compare those found by the metaheuristics. Detailed results are provided in Appendix A.

5.3.1. Exact approaches

Table 2 presents a comparison between the average results found by the formulations for each group of instances. A time limit of 1 hour was imposed for each instance. The best solution found by ILS_{CS} was provided as an initial primal bound to the solver. We report the average (Avg.), minimum (Min.) and maximum (Max.) gaps (%) found by CPLEX. Column g denotes the group, whereas column #Opt indicates the number of optimal solutions found. Finally, we also show the average CPU time, in seconds, spent by each model. The symbol "–" indicates that the corresponding model failed to solve

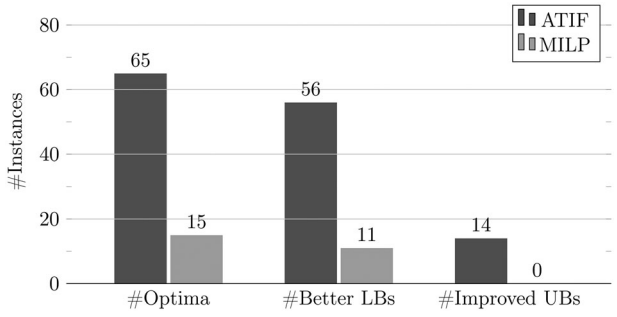


Figure 6. Number of instances where: (i) the optimal solution was found; (ii) a better lower bound was found (considering only the instances not solved to optimality); and (iii) the best upper bound provided by the metaheristics was improved.

the linear relaxation of at least one instance of the group.

The results obtained suggest that ATIF outperforms, on average, the MILP formulation by Hamzadayi and Yildiz (2017) on almost all instances. The smaller average gaps illustrate that ATIF managed to find better lower bounds for those instances where the optimal solution could not be determined. Moreover, in 14 groups (1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 16, 17 and 21), ATIF was capable of finding the optimal solution of at least one instance. Figure 6 shows the number of optimal solutions found by both formulations, where it can be observed that ATIF found a total of 65 optima, whereas the model by Hamzadayi and Yildiz (2017) could only solve 15 instances with up to 9 jobs. The same figure depicts the number of instances in which each model achieved a better lower bound (LB), considering only those where the optimal solution was not found. One can verify that the performance of ATIF is visibly superior in most cases. Moreover, ATIF could improve the best upper bound (UB) provided by the metaheuristic approaches in 14 instances, while the model from the literature could not improve any result, as also illustrated in Figure 6. Finally, ATIF also dominates their formulation when evaluating the CPU time required to solve those instances where the optimal solution was found.

Despite solving larger instances when compared to the best existing model, ATIF still could not prove the optimality of a 20-job instance, which is still a limited size. Therefore, the use of heuristic algorithms still appear to be more suitable to provide good feasible solutions for practical problems involving more jobs.

5.3.2. Metaheuristic approaches

We used the same parameters provided in Hamzadayi and Yildiz (2017) for the SA and GA algorithms. However, for a fair comparison with our algorithm, we decided to use the average CPU time obtained by ILS_{CS} on each instance as a stopping criterion for the other two methods. This enabled the GA to have a larger number of generations compared to the original number, whereas SA is restarted if the final temperature is reached before the time limit imposed.

Table 3 presents the average aggregate results found by the metaheuristic approaches. The average gaps reported are with respect to the best solution found considering all methods. We also report the CPU time, in seconds.

The results show that ILS_{CS} clearly obtained a superior performance in terms of minimum, average and maximum gaps for each group, when compared to SA and GA. Note that GA and SA failed to find the known optimal solutions in very small instances involving less than 10 jobs. Figure 7 shows the number of instances that each method found the best minimum and average objective value, respectively. It can be seen that the proposed algorithm dominates the other methods, always finding solution of better or equal quality. Overall, out of the 65 known optimal solutions, ILS_{CS} found 51 (78%), whereas SA and GA found only 28 (43%) and 12 (18%), respectively. Furthermore, the difference in performance with respect to the solution quality tends to increase with the size of the instance. In addition, ILS_{CS} appears to be more robust as the value of the gaps are more consistent than those found by SA and GA regardless of the size. It is worth mentioning that ATIF could not improve the upper bounds provided by ILS for any instance involving more than 21 jobs (see Table A1).

On average, the proposed algorithm presented a reasonable scalability, especially for instances with up to 50 jobs. Note that one should keep in mind that the move evaluation is computationally expensive ($\mathcal{O}(nm)$) and the CPU time is likely to dramatically increase with the size of the instance. Nevertheless, the average CPU time was below 150 seconds for the 100-job instances, which can be considered somewhat acceptable given the highly challenging nature of the problem.

5.4. Impact of the modified decoding algorithm

Figure 8 presents the average performance of ILS_{CS}, in terms of CPU time, for different implementations of the decoding algorithm. The graph shows the average percentage variation of the CPU time for each group of instances, which was computed with respect to the smallest CPU time among all strategies. ODA-WB corresponds to ODA with the interruption scheme, whereas MDA-NB represents the version of MDA without such scheme.

Table 3. Aggregate average results found by different metaheuristics.

g	*m*	*n*	Gap (%)									CPU (s)
			ILS_{CS}			SA			GA			
			Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	
1	2	6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.006
2		8	0.00	0.00	0.00	0.00	0.10	0.48	0.10	1.10	2.35	0.013
3		10	0.00	0.06	0.45	0.43	1.62	2.83	1.07	3.18	4.55	0.025
4		14	0.00	0.19	0.44	0.36	1.49	2.37	2.24	3.90	5.33	0.068
5		20	0.00	0.60	1.09	1.02	1.81	2.53	3.81	5.20	6.49	0.220
6	3	9	0.00	0.08	0.13	0.00	0.56	1.24	0.69	2.03	4.00	0.018
7		12	0.00	0.05	0.23	0.90	2.09	3.36	2.90	5.02	6.69	0.041
8		15	0.00	0.56	1.11	0.94	2.09	3.16	3.25	5.12	6.60	0.085
9		21	0.00	0.56	1.10	0.78	1.60	2.42	3.65	5.23	6.83	0.253
10		30	0.00	0.38	0.77	1.10	1.79	2.19	3.81	4.93	5.97	0.900
11	4	12	0.00	0.03	0.09	0.28	1.47	3.36	3.22	4.91	6.53	0.042
12		16	0.00	0.55	1.12	1.20	2.01	3.02	4.31	6.52	8.08	0.113
13		20	0.00	0.65	1.13	1.34	2.24	3.16	4.16	6.30	7.89	0.233
14		28	0.00	0.49	1.10	1.58	2.32	3.14	4.25	5.96	7.76	0.705
15		40	0.00	0.52	0.90	1.80	2.49	3.18	4.35	5.38	6.65	2.624
16	5	15	0.00	0.15	0.56	0.90	1.99	3.02	2.96	5.72	7.78	0.088
17		20	0.00	0.60	1.07	1.53	2.48	3.60	4.30	6.23	8.29	0.247
18		25	0.00	0.54	1.00	1.08	2.35	3.37	5.23	7.19	9.10	0.534
19		35	0.00	0.65	1.22	2.30	3.05	3.77	5.13	6.82	8.27	1.815
20		50	0.00	0.53	0.96	2.71	3.20	3.67	4.36	5.54	6.88	6.135
21	7	21	0.00	0.51	1.06	1.80	3.14	4.86	6.39	9.87	13.69	0.317
22		28	0.00	0.73	1.49	2.02	3.10	4.20	6.15	8.40	11.12	0.922
23		35	0.00	0.82	1.60	2.91	3.74	4.61	5.81	8.86	11.39	2.128
24		49	0.00	0.70	1.16	3.45	4.28	4.87	5.91	8.03	9.73	6.792
25		70	0.00	0.46	1.00	4.60	5.09	5.54	5.67	7.68	9.36	24.790
26	10	30	0.00	1.13	1.85	3.27	5.10	6.66	8.76	13.53	18.59	1.466
27		40	0.00	1.29	2.46	4.75	6.23	7.40	11.01	14.36	19.36	4.535
28		50	0.00	0.94	1.80	5.25	6.45	7.43	10.14	13.85	17.22	10.470
29		70	0.00	0.85	1.55	7.33	8.53	9.43	10.05	13.14	15.65	39.072
30		100	0.00	0.88	1.71	9.07	9.93	10.63	9.81	12.01	14.53	147.168

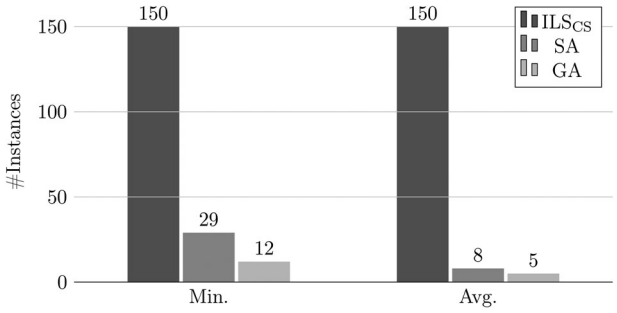


Figure 7. Number of instances where the minimum (best known) and average solution were found, respectively, by each method.

From Figure 8 we can verify that: (i) the interruption strategy yields a reduction on the average CPU time from 0 to 100% when comparing ODA with ODA-WB, and a reduction of 0 to 25% when comparing MDA with MDA-NB; (ii) for very small instances with $n/m \approx m$ it is visible that the interruption scheme becomes somewhat irrelevant; (iii) MDA reduces the average CPU time from 175 to 275%, depending on the group of instances, with respect to ODA. These results suggest that the modifications performed in the original decoding algorithm helped ILS_{CS} to achieve significant speed ups, thus dramatically increasing its runtime performance.

6. Concluding remarks

In this work, we proposed exact and metaheuristic approaches for identical parallel machine scheduling with a common server and sequence-dependent setup times. We considered the same version of the problem introduced in Hamzadayi and Yildiz (2017) where the objective is to minimize the makespan. The exact method consists of an arc-time-indexed formulation (ATIF), whereas the metaheuristic algorithm is a multi-start procedure based on iterated local search and it was here denoted as ILS_{CS}. The algorithm iterates over a job priority sequence performing swap moves and using a modified decoding algorithm (MDA), which improves the original one developed in Hamzadayi and Yildiz (2017), to compute the solution cost.

Computational experiments were carried out on 150 instances, divided into 30 groups, each with a distinct setting with respect to the number of jobs and machines. The results obtained by ATIF were compared with the mixed integer linear programming (MILP) formulation from Hamzadayi and Yildiz (2017), and those achieved by ILS_{CS} were compared with a simulated annealing (SA) and a genetic algorithm (GA), also put forward by the same authors. ATIF clearly outperformed the existing MILP model and it was capable of solving instances with up to 21 jobs to optimality, whereas the latter could only prove the optimality of instances involving up to 9 jobs. In addition, ATIF also provided, on average, better lower bounds for the unsolved instances. Likewise, ILS_{CS} also outperformed the other two approaches, finding 78% of

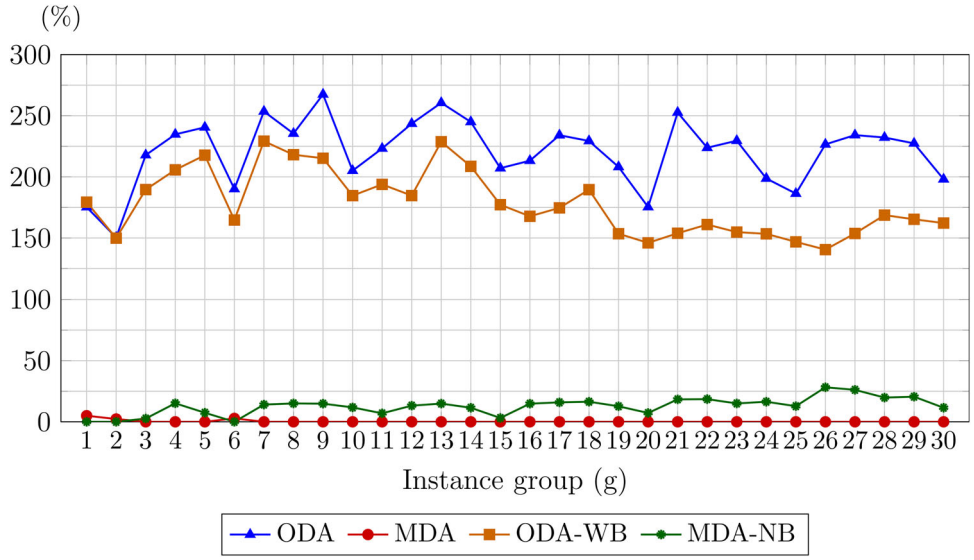


Figure 8. Average CPU time variation according to the time of instance group.

the known optimal solutions, against 43% and 18% of SA and GA, respectively. Moreover, ILS_{CS} systematically produced, on average, the best upper bounds for those instances where the optimal solutions are not available.

We also conducted experiments to illustrate the benefits of using MDA instead of the original decoding algorithm (ODA). The results show the significant speed ups achieved by ILS_{CS} when implementing the first algorithm, which in some cases was 275% faster than when using the latter procedure. Therefore, MDA played a critical role in the runtime performance of the metaheuristic.

Future work may include the development of more efficient procedures to speed up the local search phase without compromising the solution quality. One could also implement alternative perturbation mechanisms to improve the robustness of the algorithm. Finally, there is definitely room for devising enhanced exact approaches capable of consistently solving instances with more than 20 jobs.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This research was partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), grants 428549/2016-0 and 307843/2018-1, and by Comissão de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – Finance Code 001.

References

Abdekhodaee, A. H., Wirth, A., & Gan, H. S. (2004). Equal processing and equal setup time cases of scheduling parallel machines with a single server. *Computers & Operations Research*, *31*(11), 1867–1889. doi:10.1016/S0305-0548(03)00144-8

Abdekhodaee, A. H., Wirth, A., & Gan, H.-S. (2006). Scheduling two parallel machines with a single server: The general case. *Computers & Operations Research*, *33*(4), 994–1009. doi:10.1016/j.cor.2004.08.013

Balakrishnan, N., Kanet, J. J., & Sridharan, V. (1999). Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers & Operations Research*, *26*(2), 127–141. doi:10.1016/S0305-0548(98)00051-3

Gan, H.-S., Wirth, A., & Abdekhodaee, A. (2012). A branch-and-price algorithm for the general case of scheduling parallel machines with a single server. *Computers & Operations Research*, *39*(9), 2242–2247. doi:10.1016/j.cor.2011.11.007

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A.H.G.Rinnooy (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics,*, *5*, 287–326. doi:10.1016/S0167-5060(08)70356-X.

Hall, N. G., Potts, C. N., & Sriskandarajah, C. (2000). Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, *102*(3), 223–243. doi:10.1016/S0166-218X(99)00206-1

Hamzadayi, A., & Yildiz, G. (2016). Hybrid strategy based complete rescheduling approaches for dynamic m identical parallel machines scheduling problem with a common server. *Simulation Modelling Practice and Theory*, *63*, 104–132. doi:10.1016/j.simpat.2016.02.010

Hamzadayi, A., & Yildiz, G. (2017). Modeling and solving static m identical parallel machines scheduling problem with a common server and sequence dependent setup times. *Computers & Industrial Engineering*, *106*, 287–298. doi:10.1016/j.cie.2017.02.013

Hasani, K., Kravchenko, S. A., & Werner, F. (2014a). Block models for scheduling jobs on two parallel machines with a single server. *Computers & Operations Research*, *41*(1), 94–97. doi:10.1016/j.cor.2013.08.015

Hasani, K., Kravchenko, S. A., & Werner, F. (2014). Simulated annealing and genetic algorithms for the two-machine scheduling problem with a single server. *International Journal of Production Research*, *52*(13), 3778–3792. doi:10.1080/00207543.2013.874607

Huang, S., Cai, L., & Zhang, X. (2010). Parallel dedicated machine scheduling problem with sequence-dependent setups and a single server. *Computers & Industrial Engineering*, *58*(1), 165–174. doi:10.1016/j.cie.2009.10.003

Kim, M.-Y., & Lee, Y. H. (2012). MIP models and hybrid algorithm for minimizing the makespan of parallel machines scheduling problem with a single server. *Computers & Operations Research*, *39*(11), 2457–2468. doi:10.1016/j.cor.2011.12.011

Kramer, A., & Subramanian, A. (2019). A unified heuristic and an annotated bibliography for a large class of earliness–tardiness scheduling problems. *Journal of Scheduling*, *22*(1), 21–57 doi:10.1007/s10951-017-0549-6

Lourenço, H. R., Martin, O. C., & Stützle, T. (2019). Iterated local search: Framework and applications. In M. Gendreau and J.-Y. Potvin (Eds.), *Handbook of metaheuristics* (Chapter 5, pp. 129–168). Cham: Springer International Publishing.

Nesello, V., Subramanian, A., Battarra, M., & Laporte, G. (2018). Exact solution of the single-machine scheduling problem with periodic maintenances and sequence-dependent setup times. *European Journal of Operational Research*, *266*(2), 498–507. doi:10.1016/j.ejor.2017.10.020

Pessoa, A., Uchoa, E., Poggi de Aragão, M., & Rodrigues, R. (2010). Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, *2*(3/4), 259–290. doi:10.1007/s12532-010-0019-z

Silva, Y. L. T. V., Subramanian, A., & Pessoa, A. A. (2018). Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times. *Computers & Operations Research*, *90*, 142–160. doi:10.1016/j.cor.2017.09.006

Subramanian, A., Battarra, M., & Potts, C. (2014). 17). An iterated local search heuristic for the single machine total weighted tardiness problem with sequence-dependent setup times. *International Journal of Production Research*, *52*(9), 2729–2742. doi:10.1080/00207543.2014.883472

Subramanian, A., & Farias, K. (2017). Efficient local search limitation strategy for single machine total weighted tardiness scheduling with sequence-dependent setup times. *Computers & Operations Research*, *79*, 190–206. doi:10.1016/j.cor.2016.10.008

Zhang, L., & Wirth, A. (2009). On-line scheduling of two parallel machines with a single server. *Computers & Operations Research*, *36*(5), 1529–1553. doi:10.1016/j.cor.2008.02.015

Appendix A. Detailed results

Table A1. Detailed results found by different proposed approaches.

Instance	ATIF			ILS_{CS}		SA		GA		Avg
	UB	LB	CPU (s)	Best	Avg	Best	Avg	Best	Avg	CPU(s)
6n2m1	188	188	0.16	188	188.00	188	188.00	188	188.00	<0.01
6n2m2	201	201	0.13	201	201.00	201	201.00	201	201.00	<0.01
6n2m3	179	179	0.11	179	179.00	179	179.00	179	179.00	<0.01
6n2m4	151	151	0.09	151	151.00	151	151.00	151	151.00	<0.01
6n2m5	188	188	0.11	188	188.00	188	188.00	188	188.00	<0.01
8n2m1	244	244	0.75	244	244.00	244	244.00	244	245.50	<0.01
8n2m2	170	170	0.41	170	170.00	170	170.30	170	172.30	<0.02
8n2m3	245	245	0.74	245	245.00	245	245.00	245	246.50	<0.01
8n2m4	209	209	0.56	209	209.00	209	209.00	210	211.00	<0.01
8n2m5	219	219	0.67	219	219.00	219	219.70	219	223.30	<0.01
10n2m1	264	264	2.52	264	264.30	264	270.40	265	273.80	0.03
10n2m2	286	286	3.31	286	286.00	286	289.20	286	293.00	0.02
10n2m3	270	270	2.48	270	270.50	275	277.00	277	282.60	0.03
10n2m4	386	386	4.89	386	386.00	386	390.10	393	396.30	0.03
10n2m5	344	344	3.82	344	344.00	345	347.10	346	352.20	0.03
14n2m1*	467	467	326.52	467	467.50	467	471.70	473	479.60	0.06
14n2m2	388	388	32.47	388	389.80	388	396.00	397	406.40	0.07
14n2m3	402	402	465.48	402	402.80	402	405.80	414	416.90	0.07
14n2m4	503	503	50.86	503	503.00	512	514.40	516	524.70	0.07
14n2m5	444	444	118.29	444	444.80	444	449.10	453	462.00	0.07
20n2m1*	633	336.49	3600	633	637.30	638	644.30	655	667.50	0.23
20n2m2	658	350.86	3600	658	661.20	666	668.30	682	689.70	0.22
20n2m3	518	272	3600	518	520.90	523	527.90	543	549.70	0.22
20n2m4	594	315.08	3600	594	597.10	600	604.00	614	622.20	0.22
20n2m5	617	328.26	3600	617	621.50	624	630.00	640	647.10	0.22
9n3m1	207	207	0.39	207	207.00	207	208.30	209	212.20	0.02
9n3m2	163	163	0.23	163	163.00	163	165.40	166	167.40	0.02
9n3m3	151	151	0.26	151	151.60	151	151.80	152	153.60	0.02
9n3m4	151	151	0.32	151	151.00	151	151.20	151	153.90	0.02
9n3m5	227	227	0.54	227	227.00	227	227.10	227	229.90	0.02
12n3m1	256	256	2.66	256	256.60	258	261.00	267	270.50	0.05
12n3m2	269	269	2.61	269	269.00	278	278.90	278	287.00	0.04
12n3m3	206	206	10.49	207	207.00	207	209.70	209	214.20	0.04
12n3m4	271	271	2.44	271	271.10	272	276.90	280	286.30	0.04
12n3m5	271	271	3.11	271	271.00	271	274.60	278	280.80	0.04
15n3m1	285	285	63.16	285	286.30	285	290.40	293	299.20	0.09
15n3m2	370	370	14.92	370	371.00	374	376.40	378	385.20	0.09
15n3m3	329	329	120.72	329	330.40	331	335.10	342	345.60	0.09
15n3m4	297	297	13.92	297	299.30	302	304.90	307	314.20	0.08
15n3m5	303	303	12.55	303	305.70	307	310.00	315	320.20	0.08

(*continued*)

Table A1. Continued.

Instance	ATIF			ILS_{CS}		SA		GA		Avg
	UB	LB	CPU (s)	Best	Avg	Best	Avg	Best	Avg	CPU(s)
21n3m1	470	314.92	3600	470	472.10	474	477.60	488	494.80	0.26
21n3m2	483	324.86	3600	483	487.40	488	491.40	504	511.00	0.26
21n3m3	502	335.49	3600	502	503.50	504	508.00	518	527.30	0.26
21n3m4*	381	381	3040.32	387	388.60	391	393.80	400	405.80	0.25
21n3m5	512	345.71	3600	512	515.80	515	520.50	530	538.50	0.24
30n3m1	657	343.10	3600	657	658.80	662	666.90	677	688.30	0.88
30n3m2*	624	325.75	3600	624	627.00	628	634.30	653	658.30	0.94
30n3m3	574	296.10	3600	574	576.10	579	583.90	602	605.00	0.86
30n3m4	592	308.75	3600	592	593.80	603	604.00	612	621.70	0.90
30n3m5	583	305.55	3600	583	585.90	591	594.90	601	606.10	0.92
12n4m1	215	215	1.12	215	215.30	216	216.90	217	219.90	0.04
12n4m2	229	229	1.44	229	229.00	229	231.60	236	238.80	0.05
12n4m3	167	167	0.89	167	167.00	167	169.90	176	179.30	0.04
12n4m4	115	115	0.47	115	115.00	115	117.60	120	123.10	0.04
12n4m5	213	213	1.61	213	213.00	215	215.80	218	220.60	0.04
16n4m1	243	243	100.31	245	245.90	248	250.40	256	259.70	0.11
16n4m2	247	247	112.34	247	248.30	250	252.40	257	261.90	0.11
16n4m3	224	224	79.77	224	224.40	226	227.30	233	238.80	0.11
16n4m4	228	228	41.16	228	230.70	232	232.90	238	245.10	0.11
16n4m5	218	218	48.96	218	219.10	220	222.50	228	232.10	0.12
20n4m1	309	227	3600	309	310.90	314	317.10	326	333.00	0.23
20n4m2	318	318	2677.52	321	322.70	327	328.60	328	337.30	0.23
20n4m3	338	338	506.89	340	343.40	344	348.90	356	361.60	0.24
20n4m4	357	357	842.06	357	360.00	362	364.10	374	383.20	0.25
20n4m5	327	327	2323.17	330	330.90	332	335.20	342	346.30	0.22
28n4m1	402	209.51	3600	402	404.00	408	410.10	422	425.10	0.71
28n4m2	429	224	3600	429	430.80	435	438.90	446	456.60	0.71
28n4m3	416	220.24	3600	416	418.70	426	428.50	432	441.80	0.68
28n4m4*	417	217.68	3600	417	419.40	425	427.80	438	444.20	0.75
28n4m5	436	229.21	3600	436	437.30	439	443.40	451	457.40	0.68
40n4m1*	633	–	3600	633	638.30	646	651.70	663	669.60	2.68
40n4m2	627	–	3600	627	629.00	638	641.30	654	659.50	2.54
40n4m3	651	–	3600	651	654.30	663	667.20	684	689.20	2.50
40n4m4	672	–	3600	672	674.60	680	685.30	694	702.40	2.67
40n4m5	550	–	3600	550	553.00	562	565.00	574	580.60	2.73
15n5m1	187	187	9.38	187	187.20	188	190.90	195	199.70	0.09
15n5m2	187	187	18.11	187	187.00	187	188.90	194	196.20	0.09
15n5m3	169	169	3.20	169	169.30	172	173.60	174	179.50	0.09
15n5m4	168	168	2.03	168	168.60	169	170.90	169	176.80	0.08
15n5m5	187	187	2.22	187	187.20	190	191.50	193	197.20	0.09
20n5m1	244	244	349.81	245	246.10	248	250.40	256	259.10	0.27
20n5m2*	233	233	1766.41	236	236.90	238	241.40	243	250.00	0.24
20n5m3	248	248	427.93	250	252.40	255	256.90	262	267.10	0.24
20n5m4	289	289	273.15	291	292.40	296	297.40	303	306.90	0.24
20n5m5	214	214	313.34	215	216.60	219	221.30	226	230.40	0.24
25n5m1	342	234.78	3600	342	344.70	347	350.60	361	366.40	0.53
25n5m2	229	166.72	3600	229	230.20	232	237.20	244	251.70	0.54
25n5m3	357	243.71	3600	357	359.10	361	364.40	373	378.30	0.52
25n5m4	357	241.77	3600	357	358.20	359	362.50	375	379.30	0.57
25n5m5	310	211.13	3600	310	311.40	313	316.30	324	330.80	0.52
35n5m1	405	211.34	3600	405	408.70	415	419.00	431	435.30	1.84
35n5m2*	404	208.34	3600	404	406.50	413	416.00	424	431.80	1.84
35n5m3	412	212.81	3600	412	414.00	420	423.10	428	435.10	1.74
35n5m4	421	218.65	3600	421	425.30	432	434.50	446	451.60	1.88
35n5m5	402	209.23	3600	402	402.90	411	413.70	420	429.50	1.76
50n5m1	600	–	3600	600	603.00	618	621.30	632	637.90	6.69
50n5m2*	666	–	3600	666	668.70	679	684.80	690	698.10	6.22
50n5m3	656	–	3600	656	660.90	676	677.30	688	693.80	5.58
50n5m4	650	–	3600	650	653.60	668	671.20	677	684.90	6.41
50n5m5	647	–	3600	647	649.90	665	667.10	672	682.20	5.77
21n7m1	179	179	416.48	180	180.10	181	183.60	190	194.20	0.29
21n7m2	158	158	75.64	158	159.70	163	165.00	169	173.10	0.31
21n7m3	180	180	1418.52	182	182.20	183	185.40	189	199.00	0.33
21n7m4	205	205	132.73	206	206.70	209	212.00	218	225.40	0.33
21n7m5	154	154	56.70	154	155.50	159	160.90	169	174.20	0.33
28n7m1	229	175.14	3600	229	231.10	234	238.10	243	250.10	0.89
28n7m2	267	191.48	3600	267	268.60	272	275.50	279	286.20	0.91
28n7m3	290	204.08	3600	290	292.60	298	299.00	307	314.00	0.93
28n7m4	267	187.37	3600	267	268.30	272	273.50	284	287.50	0.93
28n7m5	215	165.94	3600	215	216.60	218	221.10	232	235.70	0.96
35n7m1	311	162	3600	311	313.40	319	322.30	329	338.90	2.05
35n7m2	275	183.15	3600	275	277.40	283	285.90	289	299.10	2.16
35n7m3	316	202.30	3600	316	320.30	328	330.00	343	346.90	2.16
35n7m4*	351	202.43	3600	351	353.20	361	363.10	371	380.50	2.17
35n7m5	332	173	3600	332	333.50	340	342.80	345	359.80	2.11

(*continued*)

Table A1. Continued.

Instance	ATIF			ILS_{CS}		SA		GA		Avg
	UB	LB	CPU (s)	Best	Avg	Best	Avg	Best	Avg	CPU(s)
49n7m1	435	–	3600	435	439.20	448	455.40	468	473.30	6.65
49n7m2	447	–	3600	447	448.30	463	464.60	466	478.50	6.88
49n7m3	481	–	3600	481	483.60	497	499.00	502	516.50	6.98
49n7m4	433	222.40	3600	433	436.40	449	452.80	462	468.90	6.63
49n7m5*	466	–	3600	466	470.20	483	486.80	497	506.30	6.82
70n7m1	624	–	3600	624	626.40	652	655.60	652	669.90	26.04
70n7m2	632	–	3600	632	634.10	659	662.40	669	681.50	24.59
70n7m3	624	–	3600	624	628.40	653	655.50	664	671.00	25.25
70n7m4*	620	–	3600	620	623.00	648	651.90	657	666.60	24.48
70n7m5	570	–	3600	570	572.20	599	600.70	602	616.50	23.60
30n10m1	180	143	3600	180	180.80	183	186.10	191	199.10	1.33
30n10m2	187	150	3600	187	188.50	193	196.70	205	215.10	1.56
30n10m3	182	155	3600	182	185.20	189	192.20	204	209.40	1.48
30n10m4	171	144	3600	171	172.90	177	181.30	186	195.20	1.41
30n10m5*	195	162	3600	195	198.00	203	205.30	209	220.00	1.56
40n10m1	267	180.94	3600	267	270.20	279	284.00	299	306.80	4.70
40n10m2	274	186.34	3600	274	275.90	283	287.80	302	306.50	4.52
40n10m3	258	176.16	3600	258	262.30	270	273.90	288	295.00	4.45
40n10m4*	244	166.13	3600	244	248.10	260	261.90	266	282.00	4.88
40n10m5	230	154.61	3600	230	232.80	241	244.40	258	264.80	4.14
50n10m1	316	161.96	3600	316	319.90	334	338.50	347	360.30	10.37
50n10m2	332	171.30	3600	332	336.10	350	353.00	367	373.90	10.24
50n10m3	326	166.23	3600	326	327.60	342	344.20	352	366.30	9.73
50n10m4	330	167.03	3600	330	331.80	349	351.20	368	375.00	11.20
50n10m5*	314	161	3600	314	317.80	328	335.30	348	366.10	10.81
70n10m1	435	–	3600	435	439.80	472	476.30	476	498.00	42.16
70n10m2	449	–	3600	449	451.60	480	482.70	486	499.30	38.59
70n10m3	455	–	3600	455	459.00	489	492.60	509	514.00	37.93
70n10m4	445	–	3600	445	447.70	473	480.70	490	506.70	37.42
70n10m5	415	–	3600	415	419.50	446	453.80	459	469.70	39.27
100n10m1	665	–	3600	665	672.90	731	732.20	730	742.60	147.21
100n10m2	648	–	3600	648	653.30	701	710.80	713	720.10	151.22
100n10m3	645	–	3600	645	648.80	704	710.60	713	729.80	158.42
100n10m4	653	–	3600	653	657.70	711	716.40	717	729.60	141.21
100n10m5	672	–	3600	672	679.30	734	739.10	732	755.10	137.78