# Efficient heuristics and metaheuristics for the unrelated parallel machine scheduling problem with release dates and setup times

Mohamed Elamine Athmani
mohamed_elamine.athmani@utt.fr
University of Technology of Troyes, France

Taha Arbaoui
taha.arboaui@utt.fr
University of Technology of Troyes, France

Younes Mimene
hy_mimene@esi.dz
University of Technology of Troyes, France
École nationale Supérieure d'Informatique Algiers, Algeria

Farouk Yalaoui
farouk.yaloui@utt.fr
University of Technology of Troyes, France

## ABSTRACT

Parallel machine scheduling problems are among the most studied scheduling problems in the literature. We study the unrelated parallel machine scheduling problem with release dates and machine- and sequence-dependent setup times to minimize the makespan. We introduce three heuristics, five local search methods and three metaheuristics for the problem: the Late Acceptance Hill Climbing and two variants of Simulated Annealing. Furthermore, we introduce a three-set 1620-instance benchmark in which the number of jobs and machines, release dates, processing and setup times are generated according to existing procedures. The proposed approaches are analyzed and compared on the proposed benchmark. We compare the proposed heuristics and derive the best heuristic to be used to generate the initial solution of the metaheuristics. Moreover, we show that the different metaheuristics are efficient, each performing best on one of the sets.

**ACM Reference Format:**
Mohamed Elamine Athmani, Taha Arbaoui, Younes Mimene, and Farouk Yalaoui. 2022. Efficient heuristics and metaheuristics for the unrelated parallel machine scheduling problem with release dates and setup times. In *Genetic and Evolutionary Computation Conference (GECCO '22), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3512290.3528857

## 1 INTRODUCTION

Parallel machine scheduling (PMS) problems have been a widely studied problem in the literature [26]. Several exact and heuristic approaches have been proposed to tackle them. Depending on the constraints and the objective considered, algorithms have been designed to solve multiple parallel scheduling problems with different constraints: release dates, setup times, due dates, precedence, etc. and objective functions: makespan, total completition time, total tardiness, etc. These constraints and objective functions come from several industrial applications such as printing [17], semiconductor and automobile manufacturing [14, 31] and shipbuilding [16], among others.

*GECCO '22, July 9–13, 2022, Boston, MA, USA*
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9237-2/22/07…$15.00
https://doi.org/10.1145/3512290.3528857

The makespan minimization is the most studied objective function in the literature of parallel machine scheduling problems. The unrelated parallel machine scheduling problem (UPMS) with makespan minimization, denoted $R||C_{max}$ in the notation proposed by [15], has been proved NP-hard in its simplest form $P||C_{max}$ by [22]. The problem remains NP-Hard when additional constraints such as release dates, precedence constraints, due dates and setup times are considered.

Several studies have proposed exact and heuristic algorithms for $R||C_{max}$ [25]. Recent studies have considered problems with additional constraints such as setup times [6], release dates [23], due dates [24], precedence constraints [1], resources [11] and availability constraints [20]. These constraints are often solely considered. However, recent studies combined these constraints to bridge the gap between research and practice [11, 24]. We tackle the unrelated parallel machine scheduling problem with release dates and machine- and sequence-dependent setup times with makespan minimization, denoted $R|r_j, s_{ijk}|C_{max}$. To the best of our knowledge, $R|r_j, s_{ijk}|C_{max}$ has not been considered in the literature [28]. However, several works have considered slightly different variants where additional constraints are added [31] or removed [29] and different objective functions [19, 31]. In the sequel, we discuss these variants and the proposed approaches to solve them.

The UPMS with setup times has received greater attention in the last years due to its practical importance. Several heuristic and metaheuristics approaches have been proposed. Vallada and Ruiz [29] proposed four versions of a genetic algorithm to solve $R|s_{ijk}|C_{max}$. The algorithm was based on a local-search-enhanced crossover and mutations operator. They introduced a new benchmark, comprising 1640 instances, that was randomly generated. Arnaout et al. [5] introduced a two-stage ant colony optimization algorithm for the same problem. The authors later proposed a second version of the algorithm [4] that led to improved results. It is worth noting that even though the works of Vallada and Ruiz [29] and Arnaout et al. [4, 5] tackle the same problem, they tested their approaches on different benchmarks. Terzi et al. [27] proposed a late-acceptance hill climbing approach for the same problem. Recently, Fang et al. [10] proposed a hybrid metaheuristic to solve the same problem in both benchmarks.

PMS problems with release dates have received similar attention. Chen et al. [8] proposed a Mixed Integer Program (MIP) and a hybrid tabu search approach to solve a practical PMS problem in the ion implantation process of wafer fabrication. They considered release dates, sequence-dependent setup times and expiration times to minimize three objective functions. They tested their approach on real-world data and showed that the proposed approach is efficient in attaining near-optimal solutions. Afzalirad and Rezaeian [1] tackled the PMS problem using a genetic algorithm (GA) and an artificial immune system (AIS). They considered release dates, sequence-dependent setup times, machine eligibility, precedence and resource constraints. The proposed approaches were tested on a randomly-generated benchmark where AIS is shown to be more efficient. Al-harkan et al. [2] proposed a modified harmony search algorithm for the PMS problem where release dates, setup times and resource constraints are considered. The reader is referred to the comprehensive suvery for additional works on PMS problems [3].

The literature overview shows that several works have considered different variants of PMS problems. To the best of our knowledge, $R|r_j, s_{ijk}|C_{max}$ has not been considered and there does not exist any benchmark for the problem. The contribution of this work is threefold. A benchmark comprising 1620 instances is proposed and made available for the scheduling community. Several efficient heuristics and metaheuristics for $R|r_j, s_{ijk}|C_{max}$ are proposed and analyzed. A detailed analysis shows the performance of the proposed approaches on the different classes of instances. Particularly, we show that instance characteristics impact the performance of the different approaches and hence a general conclusion cannot be drawn on the superiority of one approach over the others.

The remainder of the paper is as follows. Section 2 provides a formal description of the problem and an example instance. We describe in Section 3 the proposed heuristics and metaheuristics and provide their algorithms. Section 4 is devoted to discussing the computational results of the proposed approaches. The conclusion is to be found in Section 5.

## 2 PROBLEM DESCRIPTION

We address the UPMS problem in which the objective function is to minimize the makespan of a set of $n$ jobs to be processed by $m$ unrelated parallel machines ($R$) without preemption. Each machine can only process one job at a time, where each job $j$ has a processing time, denoted $p_{jk}$, that depends on the machine $k$ in which it is executed. There is no relationship between the speeds of the machines. We consider two constraints: release dates and machine- and sequence-dependent setup times. The release date of job $j$ is denoted $r_j$. For setup times, we consider machine- and sequence-dependent setup times between jobs inside a machine. $s_{ijk}$ denotes the time to set up job $j$ after job $i$ on machine $k$. The setup of a job $j$ on machine $k$ cannot start before its release date. The setup times between two jobs $i$ and $j$ on a machine $k$ can be different if the order of jobs is reversed ($s_{ijk} \neq s_{jik}$). The considered problem is denoted $R|r_j, s_{ijk}|C_{max}$ [26].

To illustrate the problem, we provide in Tables 1 and 2 data on an example instance that comprises 5 jobs and 2 machines. As shown in Table 1, the processing times of jobs differ from machine 1 to machine 2. Similarly, setup times are different between machine 1 and machine 2 and dependent on the sequence between jobs. It is worth noting that the diagonal values of setup times are considered

| Job | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $r_j$ | 3 | 5 | 1 | 0 | 3 |
| $p_{j1}$ | 2 | 1 | 2 | 6 | 4 |
| $p_{j2}$ | 2 | 4 | 5 | 2 | 1 |

**Table 1: Release dates and processing times of an example with two machines and five jobs**

| $s_{ij1}$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 9 | 5 | 1 | 2 | 3 |
| 2 | 7 | 3 | 5 | 1 | 6 |
| 3 | 3 | 5 | 7 | 2 | 7 |
| 4 | 5 | 3 | 3 | 1 | 2 |
| 5 | 1 | 1 | 2 | 2 | 3 |

| $s_{ij2}$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1 | 10 | 8 | 3 | 4 |
| 2 | 4 | 2 | 2 | 1 | 5 |
| 3 | 2 | 1 | 2 | 5 | 2 |
| 4 | 5 | 2 | 1 | 4 | 3 |
| 5 | 4 | 6 | 1 | 3 | 1 |

**Table 2: Setup times of an example with two machines and five jobs**

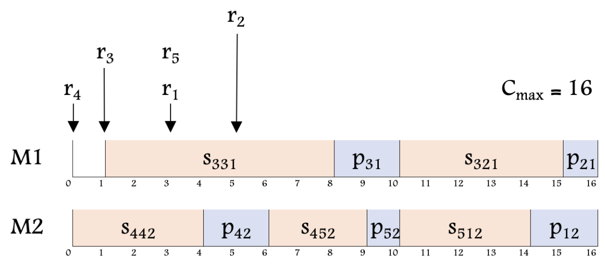

**Figure 1: A schedule solution in which jobs 2 and 3 are scheduled on the first machine and jobs 1, 4 and 5 are scheduled on the second machine. The makespan is 16.**

when the job starts on the machine. For example, if job 4 starts on the second machine, it will have a setup time equal to 4.

Figure 1 displays a feasible solution for the example instance provided in Tables 1 and 2. As shown in the figure, jobs 2 and 3 are scheduled on the first machine while jobs 1, 4 and 5 are scheduled on the second machine. For job 3 that starts on the first machine, it cannot start before time 1 as $r_3 = 1$. Moreover, the setup time that is considered for job 3 is 7 ($s_{331} = 7$). The following setup time between jobs 3 and 2 is $s_{321} = 5$. On the second machine, job 4 can start at time 0 as $r_4 = 0$ and $s_{442} = 4$. The jobs that follow are jobs 5 and 1 for which release dates are $r_5 = 3$ and $r_1 = 3$. Setup times are $s_{452} = 3$ and $s_{512} = 4$. The obtained $C_{max}$ is 16.

## 3 PROPOSED METHODS

This work introduces three efficient heuristics and three local-search-based metaheuristics for $R|r_j, s_{ijk}|C_{max}$. The heuristics are used to provide initial solutions for the metaheuristics. To compare the efficiency of the metaheuristics, all the metaheuristics use the same local search procedures. In the sequel, we describe the proposed approaches and provide their algorithms.

**Algorithm 1** Best-Insertion-Based Approach

```
function BIBA($Instance$)
    $S* \leftarrow$ EMPTY_SOL()
    $\bar{N} \leftarrow N$
    repeat
        min_ci $\leftarrow \infty$
        for i in $\bar{N}$ do
            for k in M do
                /* evaluate inserting i at the end of machine k */
                curr_ci $\leftarrow$ CI_INSERT($S*, i, k$)
                if min_ci > curr_ci then
                    min_ci $\leftarrow$ curr_ci
                    best_job, best_machine $\leftarrow$ i, k
                end if
            end for
        end for
        $\bar{N} \leftarrow \bar{N} \setminus \{best\_job\}$
        S* $\leftarrow$ S* $\cup \{best\_job, best\_machine\}$
    until $\bar{N} = \emptyset$
    return $S^*$
end function
```

**Algorithm 2** Greedy randomized adaptive search procedure

```
function GRASP($Instance$)
    $S* \leftarrow$ EMPTY_SOL()
    $\bar{N} \leftarrow N$
    repeat
        /* get a sorted insertions list */
        RCL $\leftarrow$ GENERATE_CANDIDATES(S*)
        /* get a percentage X of best insertions */
        RCL_reduced $\leftarrow$ RCL[0:X]
        /* get a random insertion */
        job, machine, pos $\leftarrow$ get_random(RCL_reduced)
        $\bar{N} \leftarrow \bar{N} \setminus \{job\}$
        S* $\leftarrow$ S* $\cup \{job, machine, pos\}$
    until $\bar{N} = \emptyset$
    return $S^*$
end function
```

## 3.1 Heuristics

We developed a set of three heuristics that can be used to construct the initial solutions. These heuristics are: BIBA, GRASP, Meta-RaPS.

*3.1.1 Best-Insertion-Based Approach (BIBA).* This approach is a greedy constructive heuristic based on consecutive calls of the best insertion method over the set of jobs $N$. This heuristic takes the list of remaining jobs to be scheduled and calls the best insertion method that assesses the insertions of each of these jobs at the current last position of each machine in the partial solution. At each iteration, the job that gives the best insertion (the best makespan) is chosen. This best insertion method is called iteratively until all the jobs are scheduled. Since the initial list of remaining jobs is ordered by job id, this method is deterministic and always returns

**Algorithm 3** Meta-heuristic for Randomized Priority Search

```
function META-RAPS($Instance$)
    $S* \leftarrow$ EMPTY_SOL()
    $\bar{N} \leftarrow N$
    repeat
        /* get a sorted insertions list */
        RCL $\leftarrow$ GENERATE_CANDIDATES($S*$)
        /* get a percentage X of best insertions */
        RCL_reduced $\leftarrow$ RCL[0:X]
        if RANDOM() $\leq p$ then
            /* get a random insertion */
            job, machine, pos $\leftarrow$ get_random(RCL_reduced)
        else
            /* get the best insertion */
            job, machine, pos $\leftarrow$ get_best(RCL_reduced)
        end if
        $\bar{N} \leftarrow \bar{N} \setminus \{job\}$
        S* $\leftarrow$ S* $\cup \{job, machine, pos\}$
    until $\bar{N} = \emptyset$
    return $S^*$
end function
```

the same solution for the same instance. Algorithm 1 describes the pseudo-code of the approach.

*3.1.2 Greedy Randomized Adaptive Search Procedure (GRASP).* The second heuristic, GRASP [13], is a random-greedy constructive algorithm where the jobs are scheduled iteratively one by one in one of the m machines. Given a partial solution $S*$, GRASP creates a list of insertions called RCL (Restricted Candidate List). Next, it assesses the insertion of each remaining job into each machine and in each position. Once this list is created, a percentage X of the best insertions is selected. A job is then randomly chosen from this reduced list and the insertion is applied. The corresponding job is considered assigned and this process is repeated until all the remaining jobs are scheduled. Algorithm 2 describes the pseudo-code of the approach. Default value of $X$ is 0.2.

*3.1.3 Meta-heuristic for Randomized Priority Search (Meta-RaPS).* Meta-RaPS [9] is also a random-greedy constructive algorithm which resembles to GRASP with the difference of having an extra parameter $p$ which is the probability of taking the best insertion or a random one from the RCL. First, the procedure builds the RCL and sorts it. It then takes the best insertion with a probability of $p$ and randomly chooses from the RCL like GRASP with a probability of $(1 - p)$. Setting the parameter p will determine the compromise between imitating GRASP ($p$ -> 0) or being greedy ($p$ -> 1). Algorithm 3 describes the pseudo-code of the approach. Default parameters are $p = 0.2, X = 0.2$.

## 3.2 Local search operators

A local search procedure is developed using multiple operators in order to improve an existing solution of the problem. This procedure is an exploration of the different neighborhoods at a certain iteration. It mainly relies on 3 basic operations:

(1) Internal swap: Exchanging the positions of two tasks inside the same machine.
(2) External swap: Exchanging the positions of two tasks from two different machines.
(3) External insertion: Moving a job from one machine to another.

Using these basic operations, the following 5 operators are used to build our local search procedure.

*3.2.1 Bottleneck internal swap.* The aim of this operator is to improve the completion time of bottleneck machines. The internal swap operation is applied on each pair of jobs $i$ and $j$ inside the bottleneck machines $k$ ($C_k = C_{max}$). The swap operation that yields the best reduction of the makespan is accepted, if it exists.

*3.2.2 Bottleneck external Insertion.* The aim of this operator is to reduce the load on bottleneck machines by reducing the number of jobs. This operator is applied to each bottleneck machine. First, a random non-bottleneck machine is selected. Second, we apply the pairs of external insertions of each job from the bottleneck machine into each position on the non-bottleneck machine. We finally accept the best insertion that either reduces the makespan or keeps it the same while reducing the completion time of the bottleneck machine.

*3.2.3 Bottleneck external swap.* The aim of this operator is to reduce the completion time of bottleneck machines by getting shorter jobs from other machines. This operator is also applied to each bottleneck machine. First, a random non-bottleneck machine is selected. Second, we apply the swap of each job of the bottleneck machine with each job of the non-bottleneck machine. We finally accept the best swap that either reduces the makespan or keeps it the same while reducing the completion time of one of the two machines.

*3.2.4 Balancing.* The aim is to balance the load of machines in terms of completion time. The procedure applies the external insertions of the last job of a bottleneck machine into each position of the non-bottleneck machines. It applies the best insertion that reduces the makespan of the solution and re-iterates until no change occurs.

*3.2.5 Inter machine insertion.* The aim is to find a better position for the tasks on each machine. We iterate between each couple of machines $k$ and $h$ and apply the external insertion on each job in the first machine $k$ into each position of the second machine $h$. We apply the best move that reduces completion times the most, and satisfies Equation (1), if it exists. Therefore, we apply at most $M \times M$ moves.

$$C_k - C_{k-new} > C_{h-new} - C_h \text{ and } C_{max-new} \leq C_{max} \qquad (1)$$

## 3.3 Metaheuristics

*3.3.1 Late Acceptance Hill Climbing metaheuristic (LAHC).* LAHC was first proposed by Burke and Bykov [7] and is an extension of the local search Hill climbing algorithm. It uses a list to save and exploit the history of the search process by saving the cost of the solution of each iteration in a history list called $H$. The algorithm is an iterative search process that uses a neighborhood generation method in order to generate a solution in the neighborhood at each iteration. The generation method is composed of two operators that are applied alternatively with a uniform random probability.

- **Random internal swap**: a machine is randomly chosen and a swap is operated between two randomly-chosen jobs.
- **Random external swap**: two machines are randomly chosen and a swap is operated between two random jobs, each from one of the machines.

Once a new solution is generated, it is only accepted if it respects one of the two conditions: a) the new solution is better than the current one; b) the new solution is better than the solution found before $LH$ iterations. The history list allows the algorithm to accept solutions that deteriorate the current solution's quality when the new solution is better than it was in previous iterations (the size of the history list is $LH$). This allows LAHC to escape local minima and explore other areas in the search space. This method has only one parameter which is the length of the history list $LH$. Algorithm 4 provides the pseudo-code.

**Algorithm 4** Late Acceptance Hill Climbing

```
function LAHC(Instance)
    $P \leftarrow 0$
    $i \leftarrow 1$
    $best\_sol \leftarrow$ INIT()
    $curr\_sol \leftarrow$ best_sol
    while $i \leq Nb_{iter}$ and $P \leq Non\_improv$ do
        $next\_sol \leftarrow$ NEIGHBOUR($curr\_sol$)
        $next\_sol \leftarrow$ LOCAL_SEARCH($next\_sol$)
        if CMAX($curr\_sol$) $\leq$ CMAX($next\_sol$)
            OR CMAX($curr\_sol$) $\leq H[i\%LH]$ then
            $curr\_sol \leftarrow next\_sol$
        end if
        if CMAX($curr\_sol$) $\leq$ CMAX($best\_sol$) then
            $best\_sol \leftarrow curr\_sol$
            $P \leftarrow 0$
        end if
        $H[i\%LH] \leftarrow$ CMAX($next\_sol$)
        $i \leftarrow i + 1$
        $P \leftarrow P + 1$
    end while
    return $best\_sol$
end function

function NEIGHBOUR($sol$)
    if RANDOM() < 0.5 then
        $next\_sol \leftarrow$ INTERNAL_RANDOM_SWAP($sol$)
    else
        $next\_sol \leftarrow$ EXTERNAL_RANDOM_SWAP($sol$)
    end if
    return $next\_sol$
end function
```

**Algorithm 5** Simulated annealing neighborhood variant 1

```
function NEIGHBOUR_1($sol$)
    $random\_number \leftarrow$ RANDOM()
    if $random\_number \leq \frac{1}{3}$ then
        $next\_sol \leftarrow$ INTERNAL_RANDOM_SWAP($sol$)
    else if $random\_number \leq \frac{2}{3}$ then
        $next\_sol \leftarrow$ EXTERNAL_RANDOM_SWAP($sol$)
    else
        $next\_sol \leftarrow$ INTER_MACHINE_INSERTION($sol$)
    end if
    return $next\_sol$
end function
```

**Algorithm 6** Simulated annealing neighborhood variant 2

```
function NEIGHBOUR_2($sol$)
    if RANDOM() $< \frac{1}{2}$ then
        $next\_sol \leftarrow$ RESTRICTED_SWAP($sol$)
    else
        $next\_sol \leftarrow$ RESTRICTED_INSERT($sol$)
    end if
    return $next\_sol$
end function
```

**Algorithm 7** Simulated annealing

```
function SimulatedAnnealing($Instance$)
    $T \leftarrow T_{max}$
    $N \leftarrow 0$
    $best\_sol \leftarrow$ INIT()
    $curr\_sol \leftarrow$ best_sol
    while $T > T_{min}$ and $N \leq Non\_improv$ do
        $i \leftarrow 0$
        while $i \leq n\_iter$ do
            $next\_sol \leftarrow$ NEIGHBOUR($curr\_sol$)
            $next\_sol \leftarrow$ LOCAL_SEARCH($next\_sol$)
            $\Delta C \leftarrow$ Cmax($curr\_sol$) − Cmax($next\_sol$)
            if $\Delta C \geq 0$ then
                $curr\_sol \leftarrow next\_sol$
            else if RANDOM() $< e^{\frac{\Delta C}{k*T}}$ then
                $curr\_sol \leftarrow next\_sol$
            end if
            if Cmax($curr\_sol$) $\leq$ Cmax($best\_sol$) then
                $best\_sol \leftarrow curr\_sol$
                $N \leftarrow 0$
            end if
            $N \leftarrow N + 1$
            $i \leftarrow i + 1$
        end while
        $T \leftarrow T * b$
    end while
    return $best\_sol$
end function
```

*3.3.2 Simulated annealing (SA).* SA is a widely used optimization metaheuristics inspired by the process of the crystallization cooling. It has shown great success in many combinatorial optimization problems [21]. The algorithm has three main parameters: the initial temperature $T_0$, the final temperature $T_f$ and the cooling factor $b$. The algorithm starts with constructing an initial solution using a heuristic algorithm. It then iteratively generates a new solution in the neighborhood using dedicated neighborhood exploration operators and decides whether to accept the newly generated solution. To accept a new solution, SA compares the quality of the new solution and the incumbent solution. In the case of a minimization problem, if the value of the new solution is smaller than the incumbent one, it is automatically accepted. Otherwise, it is also possible to accept a solution that deteriorates the quality of the incumbent solution in order to escape local minima, according to the following probability: $e^{\frac{\Delta E}{KT}}$ where $\Delta E = C_{max}^{old} - C_{max}^{new}$. This probability depends on the difference between the values, where the probability is bigger for solutions that are close to the incumbent solution. It also depends on the current temperature $T$.

The temperature $T$ is reduced during the search from $T_0$ until it reaches the final temperature $T_f$. The reduction happens each $n\_iter$ by a cooling factor $b$. During the first iterations, when the temperature is high, the probability of accepting deteriorating solutions is also high to allow exploration. As the iterations go by, temperature $T$ is reduced and the probability of accepting deteriorating solutions is lower to allow exploitation. Algorithm 7 describes the pseudo-code of the approach.

As the impact of the neighborhood operators is considerable on SA, two variants of SA are developed. The algorithm's structure is kept unchanged and the neighborhood generation operators are changed. The two variants are described below.

**First variant of SA (S1).** In the first variant of SA, 3 neighborhood operators are used to generate a solution each time. The generation procedure chooses one of the operators according to a uniform random probability. The first two operators are random internal swap and random external swap (see the previous section). A third operator which is inter machine insertion described in the local search section.

**Second variant of SA (S2).** In the first variant of SA, the neighborhood exploration concerns all machines in the solution at each iteration. Most of these moves will not improve the solution, particularly those considering random machines, since they do not necessarily include a bottleneck machine. In the second variant, we use a restricted search strategy called RSA inspired by [32]. We then limit the neighborhood generation operators to moves that will improve the current solution. These will necessarily include one of the bottleneck machines, and another non-bottleneck machine. These neighborhood operators are:

(1) Restricted swap: swaps between two jobs, the first one is from a bottleneck machine and another from a non-bottleneck machine.
(2) Restricted insert: removes a task from a bottleneck machine and inserts it into a non-bottleneck machine.

*3.3.3 Parameter tuning.* In order to determine the best generic parameters instances, ParamILS [18], a widely used in the literature to tune different kinds of algorithms (metaheuristics, solvers, etc),

| | # Instances | # Jobs | # Machines | Setup time ranges | Release factor |
|---|---|---|---|---|---|
| Small | 640 | {10,20,30,40} | {2,4,6,8} | {U(1, 9), U(1, 49), U(1, 99), U(1,124)} | {0.2, 0.6, 1.0, 1.4, 1.8} |
| Medium | 180 | {40} | {2} | U(50,100) | {0.2, 0.6, 1.0, 1.4, 1.8} |
| | | {60, 80} | {2, 4} | | |
| | | {100} | {2, 4, 6} | | |
| | | {120} | {2, 4, 6, 8} | | |
| Large | 800 | {200,400,600,800,1000} | {2,4,6,8} | {U(1, 9), U(1, 49), U(1, 99), U(1,124)} | {0.2, 0.6, 1.0, 1.4, 1.8} |

**Table 3: The proposed benchmark size and generation protocol.**

was used. ParamILS is used on a reduced benchmark of 32 instances. The reduced benchmark is chosen so as to be representative of the full benchmark while comprising a limited number of instances. It is divided into 24 training instances and 8 test instances. The objective is to get the best makespan and the stochasticity of the algorithms is taken into account.

For LAHC, we used ParamILS in order to fix the only parameter the method relies on, which is the size of the list $LH$. The fixed search range is {10, 20, 30, 40, 50} with a default value of 20. ParamILS suggested 30 as the best value. For SA, ParamILS was used on the first variant S1 and the same values were used for S2 for fair comparison. The considered parameters are: Initial temperature $T_0$ : {1.1, 1.2, 1.3, 1.4, 1.5} with a default value of 1.5; the final temperature $T_f$ : {0.01, 0.02, 0.03, 0.04, 0.05} with the default value of 0.02; and the cooling factor b: {0.99, 0.98, 0.97, 0.96, 0.95} with a default value of 0.98. These ranges create 220 configurations. The returned configuration by ParamILS is $T_0 = 1.4, T_f = 0.01$ and $b = 0.99$. Using these parameters, SA will operate 500 iterations. The rest of parameters are set as follows: $n\_iter = 20, k = 0.1$.

## 4 RESULTS

All the algorithms presented in this work were coded in Python 3.9 and ran using a PyPy interpreter on a dedicated server with Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz and 96GB of RAM. No parallelization was used during the execution of the algorithms.

The performance of each algorithm was evaluated by running extensive experiments using a benchmark of 1620 instances divided into 3 sets: small, medium and large instances generated according to the protocol described in the next section. All the methods are run 5 times and the median of the returned $C_{max}$ is considered for each instance.

### 4.1 Benchmark description

Fanjul-Peyro et al. [12] generated 1620 instances for the problem $R|s_{ijk}|C_{max}$ which includes processing times and setup times. All processing times are generated according to a random uniform distribution of $U(1, 99)$. Setup times are also generated according to a random uniform distribution with different intervals. Based on these instances, a new benchmark in which release dates are added to these 1620 instances is generated. The release dates are generated according to the protocol of [30]. Release dates are sampled from a uniform distribution $U(1, L)$, where $L$ is computed as $L = \frac{n \times \rho \times R_f}{m}$, where $n$ is the number of jobs in the instance, $m$ is the number of machines, $\rho = \frac{1}{nm} \sum_{j=1}^{n} \sum_{k=1}^{m} p_{jk}$ is the expected processing time and $R_f$ is the release factor to control the distribution of release dates. Therefore, each instance is characterized by 4 factors: the

| | | RPD (%) | | | Best count | | |
|---|---|---|---|---|---|---|---|
| N | M | BIBA | GR | MR | BIBA | GR | MR |
| 10 | 2 | 3.20 | 3.29 | **2.16** | **21** | 09 | 14 |
| | 4 | **1.81** | 2.71 | 1.97 | **22** | 13 | 16 |
| | 6 | 2.03 | **1.98** | 2.20 | **27** | 12 | 20 |
| | 8 | 4.87 | **1.72** | 1.91 | **19** | 18 | 18 |
| 20 | 2 | 2.61 | 3.65 | **2.41** | **25** | 04 | 15 |
| | 4 | **0.79** | 3.60 | 2.31 | **34** | 03 | 08 |
| | 6 | **0.09** | 3.34 | 2.97 | **37** | 00 | 6 |
| | 8 | **0.39** | 2.37 | 1.57 | **31** | 03 | 8 |
| 30 | 2 | **1.81** | 3.54 | 2.28 | **27** | 03 | 11 |
| | 4 | **0.55** | 3.98 | 2.37 | **36** | 00 | 6 |
| | 6 | **0.14** | 3.08 | 2.75 | **36** | 02 | 04 |
| | 8 | **0.00** | 3.18 | 2.25 | **38** | 00 | 03 |
| 40 | 2 | **1.06** | 3.53 | 2.30 | **30** | 02 | 08 |
| | 4 | **0.4** | 3.18 | 1.90 | **32** | 02 | 07 |
| | 6 | **0.00** | 3.44 | 2.85 | **39** | 01 | 01 |
| | 8 | **0.02** | 2.65 | 2.29 | **37** | 02 | 02 |
| Avg | | **1.24** | 3.08 | 2.28 | **31** | 05 | 09 |

**Table 4: Heuristics' metrics on the small set.**

number of jobs $n$, the number of machines $m$, the setup times range and the release date factor $R_f$.

In the small set, the factors are $n \in \{10, 20, 30, 40\}$, $m \in \{2, 4, 6, 8\}$ and setup times ranges are $\{U(1, 9), U(1, 49), U(1, 99), U(1, 124)\}$, for each combination 10 instances are generated by the authors in [12]. Release dates are added with the following factors $R_f \in \{0.2, 0.6, 1.0, 1.4, 1.8\}$ where each factor is applied to 2 of the 10 instances. This results in $4 \times 4 \times 4 \times 10 = 640$ instances.

In the medium set, for $n \in \{40, 60, 80, 100, 120\}$, the number of machines considered depends on the number of jobs. For $n = 40$, $m = 2$; for $n \in \{60, 80\}$, $m \in \{2, 4\}$; for $n = 100$, $m \in \{2, 4, 6\}$ and for $n = 120$, $m \in \{2, 4, 6, 8\}$. Setup times are generated in $U(50, 100)$ and 15 instance are generated for each combination, the release dates factors are the same as before but each factor is applied to 3 of the 15 instances. This results in 180 instances.

In the large set, the factors are $n \in \{200, 400, 600, 800, 1000\}$, $m \in \{2, 4, 6, 8\}$ and setup times and release dates ranges are the same as the small set. This results in 10*5*4*4= 800 instances. Table 3 shows the summary of the benchmark. The benchmark and the detailed results are available at www.scheduling.cc.

### 4.2 Comparing the proposed heuristics

Tables 4, 5 and 6 present the results obtained by the proposed three heuristics: BIBA, GRASP (GR) and Meta-RaPS (MR). The results

| | | RPD (%) | | | Best count | | |
|---|---|---|---|---|---|---|---|
| N | M | BIBA | GR | MR | BIBA | GR | MR |
| 40 | 2 | **0.72** | 3.80 | 3.38 | **11** | 02 | 02 |
| 60 | 2 | **0.15** | 4.06 | 2.77 | **12** | 00 | 03 |
| | 4 | **0.05** | 2.96 | 2.84 | **13** | 00 | 02 |
| 80 | 2 | **0.25** | 4.86 | 3.2 | **12** | 00 | 03 |
| | 4 | **0.01** | 4.19 | 3.28 | **14** | 00 | 01 |
| 100 | 2 | **0.08** | 4.47 | 3.08 | **13** | 00 | 02 |
| | 4 | **0.00** | 4.34 | 3.25 | **15** | 00 | 00 |
| | 6 | **0.00** | 3.51 | 2.21 | **15** | 00 | 00 |
| 120 | 2 | **0.04** | 4.87 | 3.21 | **14** | 00 | 01 |
| | 4 | **0.00** | 3.86 | 3.06 | **15** | 00 | 00 |
| | 6 | **0.00** | 3.41 | 2.69 | **15** | 00 | 00 |
| | 8 | **0.00** | 3.23 | 2.19 | **15** | 00 | 00 |
| Avg | | **0.11** | 3.96 | 2.93 | **14** | 00 | 01 |

**Table 5: Heuristics’ metrics on the medium set.**

| | | RPD (%) | | | Best count | | |
|---|---|---|---|---|---|---|---|
| N | M | BIBA | GR | MR | BIBA | GR | MR |
| 200 | 2 | **1.09** | 3.94 | 3.1 | **32** | 00 | 08 |
| | 4 | **0.00** | 3.02 | 2.32 | **40** | 00 | 00 |
| | 6 | **0.00** | 2.74 | 2.17 | **40** | 00 | 01 |
| | 8 | **0.00** | 2.31 | 1.74 | **40** | 00 | 00 |
| 400 | 2 | **1.12** | 4.47 | 3.33 | **32** | 00 | 08 |
| | 4 | **0.00** | 3.33 | 2.65 | **40** | 00 | 00 |
| | 6 | **0.00** | 3.15 | 2.04 | **40** | 00 | 00 |
| | 8 | **0.00** | 2.11 | 1.95 | **40** | 00 | 00 |
| 600 | 2 | **1.21** | 4.77 | 3.59 | **32** | 00 | 08 |
| | 4 | **0.00** | 3.53 | 2.73 | **40** | 00 | 00 |
| | 6 | **0.00** | 2.87 | 2.18 | **40** | 00 | 00 |
| | 8 | **0.00** | 2.27 | 1.92 | **40** | 00 | 00 |
| 800 | 2 | **1.23** | 4.67 | 3.71 | **32** | 00 | 08 |
| | 4 | **0.01** | 3.85 | 3.14 | **39** | 00 | 01 |
| | 6 | **0.00** | 2.88 | 2.5 | **40** | 00 | 00 |
| | 8 | **0.00** | 2.14 | 1.69 | **40** | 00 | 00 |
| 1000 | 2 | **1.31** | 5.01 | 3.92 | **32** | 00 | 08 |
| | 4 | **0.00** | 3.86 | 3.12 | **40** | 00 | 00 |
| | 6 | **0.00** | 3.23 | 2.68 | **40** | 00 | 00 |
| | 8 | **0.00** | 2.27 | 1.75 | **40** | 00 | 00 |
| Avg | | **0.30** | 3.32 | 2.61 | **38** | 00 | 02 |

**Table 6: Heuristics’ metrics on the large set.**

are aggregated by couples of $n \times m$. For the small and large set, each row represents 40 instances whereas for the medium set each row represents 15 instances. The last row presents the average (Avg) performance of each heuristic on the whole set. The first two columns provide the considered $n \times m$. The next three columns, denoted RPD, provide the average relative percentage deviation for each method. The RPD of a method $h$ on an instance is given by $RPD_h = \frac{C^h_{max} - C^{best}_{max}}{C^{best}_{max}}$ where $C^{best}_{max}$ represents the best makespan obtained by the three heuristics (BIBA, GRASP and Meta-RaPS). The next three columns, denoted Best Count, provide, for each approach, the number of instances for which the approach obtained the best solution among the three approaches. It is worth noting that two or more approaches can reach the best solution and therefore the instance is counted for both.

| | | | LA | | | S1 | | | S2 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | N | M | BIBA | GR | MR | BIBA | GR | MR | BIBA | GR | MR |
| Small | 10 | 2 | **0.00** | **0.00** | 0.01 | **0.00** | 0.16 | 0.24 | **0.00** | 0.01 | 0.01 |
| | | 4 | **0.00** | **0.00** | **0.00** | 0.05 | 0.05 | **0.00** | 0.09 | 0.05 | **0.00** |
| | | 6 | **0.00** | **0.00** | **0.00** | 0.18 | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| | | 8 | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| | 20 | 2 | **0.00** | 0.03 | 0.04 | **0.05** | 0.17 | 0.16 | 0.21 | **0.10** | 0.17 |
| | | 4 | **0.00** | **0.00** | **0.00** | 0.12 | **0.02** | 0.17 | **0.03** | 0.15 | 0.13 |
| | | 6 | **0.00** | **0.00** | **0.00** | 0.13 | **0.06** | 0.21 | 0.12 | 0.15 | **0.09** |
| | | 8 | **0.00** | **0.00** | **0.00** | 0.01 | **0.00** | **0.00** | 0.07 | 0.13 | **0.00** |
| | 30 | 2 | **0.05** | 0.11 | 0.07 | 0.26 | 0.19 | **0.14** | 0.18 | 0.24 | **0.14** |
| | | 4 | 0.02 | 0.05 | **0.01** | **0.06** | 0.07 | 0.09 | 0.10 | 0.09 | **0.06** |
| | | 6 | 0.05 | 0.05 | **0.00** | **0.00** | 0.06 | 0.06 | **0.01** | 0.08 | 0.09 |
| | | 8 | 0.02 | **0.00** | **0.00** | 0.03 | **0.00** | 0.02 | 0.05 | **0.00** | 0.06 |
| | 40 | 2 | **0.03** | 0.06 | 0.09 | 0.26 | **0.09** | 0.15 | 0.20 | **0.14** | **0.14** |
| | | 4 | 0.05 | 0.05 | **0.03** | 0.02 | **0.01** | 0.04 | 0.09 | **0.07** | 0.19 |
| | | 6 | **0.01** | 0.02 | **0.01** | **0.01** | **0.01** | 0.03 | 0.04 | 0.03 | **0.01** |
| | | 8 | 0.02 | **0.00** | 0.02 | **0.01** | **0.01** | **0.01** | **0.01** | **0.01** | 0.02 |
| | Avg | | **0.02** | **0.02** | **0.02** | 0.07 | **0.06** | 0.08 | **0.07** | 0.08 | **0.07** |
| Medium | 40 | 2 | 0.09 | 0.09 | **0.05** | 0.09 | 0.15 | **0.07** | 0.09 | **0.08** | 0.10 |
| | 60 | 2 | **0.08** | 0.26 | 0.18 | 0.10 | **0.07** | 0.12 | 0.11 | **0.03** | 0.11 |
| | | 4 | **0.03** | 0.07 | 0.04 | 0.07 | 0.02 | **0.01** | **0.06** | 0.10 | 0.10 |
| | 80 | 2 | **0.01** | 0.32 | 0.32 | 0.05 | **0.03** | 0.05 | 0.11 | **0.09** | **0.09** |
| | | 4 | 0.03 | **0.02** | 0.04 | **0.00** | 0.05 | 0.02 | **0.02** | 0.05 | 0.03 |
| | 100 | 2 | **0.04** | 0.44 | 0.43 | 0.10 | 0.09 | **0.04** | 0.09 | **0.06** | 0.09 |
| | | 4 | **0.00** | 0.05 | 0.05 | **0.04** | 0.05 | **0.02** | 0.05 | 0.05 | **0.03** |
| | | 6 | **0.01** | 0.02 | **0.01** | **0.00** | 0.05 | 0.06 | **0.02** | 0.05 | 0.19 |
| | 120 | 2 | **0.02** | 0.45 | 0.36 | 0.08 | **0.06** | 0.07 | **0.04** | 0.07 | 0.09 |
| | | 4 | **0.00** | 0.11 | 0.12 | **0.02** | 0.06 | 0.06 | **0.01** | 0.08 | 0.15 |
| | | 6 | **0.00** | 0.13 | 0.15 | **0.03** | 0.07 | 0.09 | **0.00** | 0.19 | 0.16 |
| | | 8 | **0.00** | 0.21 | 0.25 | **0.05** | 0.07 | 0.11 | **0.00** | 0.18 | 0.06 |
| | Avg | | **0.03** | 0.18 | 0.17 | **0.05** | 0.06 | 0.06 | **0.05** | 0.09 | 0.10 |
| | Avg | | **0.02** | 0.06 | 0.05 | 0.07 | **0.06** | 0.08 | **0.07** | 0.08 | 0.08 |

**Table 7: Impact of the heuristic used to generate the initial solution in terms of RPD (%) for each metaheuristic.**

As it can be seen on the tables of the three sets, the proposed BIBA heuristic outperforms the other two approaches. For the small set, GRASP and Meta-RaPS are competitive for $n = 10$ and $n = 20$. However, for the medium and large sets, BIBA obtains the best RPD and most of the best instances, except for some instances where $m = 2$.

## 4.3 Impact of the initial solution

The initial solution of a metaheuristic is known to greatly impact its performance. The proposed heuristics were used to obtain the initial solutions for the three metaheuristics (LA for LAHC, S1 and S2 for variant 1 and 2 of SA). The results presented the previous section pushed us to conduct a comparison to study their impact and to decide the best heuristic to use for every metaheuristic.

To do so, the metaheuristics’ initial solutions were generated with each one the three heuristics and executed on small and medium sets. Table 7 shows the results of RPD of each method inside each group aggregated with average by $n \times m$.

| | | RPD (%) | | | Best count | | | Time to best ratio (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| N | M | LA | S1 | S2 | LA | S1 | S2 | LA | S1 | S2 |
| 10 | 2 | **0.00** | 0.09 | 0.09 | **40** | 39 | 39 | **00.3** | 00.5 | 00.6 |
| | 4 | **0.00** | 0.06 | 0.06 | **40** | 38 | 38 | **00.2** | 00.7 | 01.3 |
| | 6 | 0.02 | 0.18 | **0.00** | 39 | 39 | **40** | **00.1** | 00.5 | 01.4 |
| | 8 | **0.00** | **0.00** | **0.00** | **40** | **40** | **40** | **00.1** | 00.3 | 00.2 |
| 20 | 2 | **0.00** | 0.49 | 0.38 | **40** | 30 | 30 | 11.5 | **05.3** | 07.4 |
| | 4 | **0.01** | 0.20 | 0.10 | **38** | **38** | **38** | 03.2 | 04.7 | **01.5** |
| | 6 | **0.00** | 0.09 | 0.16 | **40** | 38 | 35 | 04.0 | 03.3 | **02.1** |
| | 8 | **0.00** | **0.00** | **0.00** | **40** | **40** | **40** | 00.4 | 01.3 | **00.6** |
| 30 | 2 | **0.04** | 0.28 | 0.21 | **33** | 30 | **33** | 13.4 | **10.3** | 16.1 |
| | 4 | **0.09** | 0.03 | 0.10 | 35 | **37** | **37** | **04.7** | 07.2 | 06.9 |
| | 6 | 0.05 | **0.00** | **0.00** | 39 | **40** | **40** | 03.5 | 05.5 | **01.7** |
| | 8 | 0.04 | 0.02 | **0.00** | 37 | 38 | **40** | 03.8 | **02.3** | 04.1 |
| 40 | 2 | 0.24 | **0.16** | **0.16** | 29 | 30 | **31** | 16.1 | 14.6 | **12.2** |
| | 4 | 0.20 | **0.02** | 0.06 | 32 | **36** | 34 | **07.8** | 09.0 | 08.3 |
| | 6 | 0.03 | **0.01** | 0.04 | 36 | **38** | **38** | **03.2** | 08.1 | 03.1 |
| | 8 | 0.06 | 0.01 | **0.00** | 37 | 39 | **40** | **02.0** | 04.1 | 02.4 |
| Avg | | **0.05** | 0.10 | 0.09 | **37** | **37** | **37** | 04.7 | 04.9 | **04.4** |

**Table 8: Performance comparison of the proposed metaheuristics on the small set.**

For the small set, the methods' performances are comparable. BIBA and Meta-RaPS perform the best with LAHC and S2, whereas GRASP performs best with LAHC and S1. It is worth noting that none of the methods is dominating any classes. The medium set allows to differentiate between the heuristics with a clear advantage for BIBA performing best with all metaheuristics, particularly with LAHC as RPD =0.03% compared to 0.18% and 0.17% for GRASP and Meta-RaPS respectively. Tests were not performed for large instances as the trend resembles the results of the previous section. Therefore, BIBA is chosen as the default heuristic to obtain initial solutions for the proposed metaheuristics.

## 4.4 Comparing the proposed metaheuristics

Tables 8, 10 and 9 present the results obtained by the proposed metaheuristics on the small, medium and large sets respectively. Results are aggregated by classes $n \times m$ as in the previous tables. For a fair comparison, a time limit that depends on the number of jobs and machines given by the formula $Time\_limit = \frac{N*M}{2}$ was imposed to each metaheuristic. We consider three metrics: RPD, best count and time to best ratio. Time to best ratio represents the ratio of the time needed to find the best solution with respect to the total runtime of the metaheuristic. The last line of each Table represents the average over the whole set.

For the small set, overall LAHC is performing better than the two variants of SA ($RPD = 0.05$) while reaching its best solution in less than 5% of the runtime. We notice that its performance deteriorate when the number of jobs increases. The two variants of SA are comparable achieving around 0.10% in terms of RPD. All methods reach the solution fast within 5% of their runtime in average.

For the medium set, we see a clear dominance of S1 in terms of RPD and best count where it reached the best results in all classes. LAHC's performance continues to degrade when the number of jobs increases but it is the fastest to reach its best solution with 9.8%. With 25.2% time to best ratio S1 is the best method on this set.

| | | RPD (%) | | | Best count | | | Time to best ratio (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| N | M | LA | S1 | S2 | LA | S1 | S2 | LA | S1 | S2 |
| 200 | 2 | 0.17 | **0.02** | 0.09 | 26 | **32** | 29 | **26.3** | 32.2 | 33.3 |
| | 4 | 0.26 | **0.01** | 0.03 | 17 | 31 | **37** | 19.7 | 23.1 | **14.2** |
| | 6 | 0.02 | **0.00** | **0.00** | 30 | 37 | **40** | 16.2 | 07.1 | **03.8** |
| | 8 | 0.01 | 0.01 | **0.00** | 32 | 35 | **39** | 16.5 | 05.9 | **02.8** |
| 400 | 2 | 0.34 | 0.04 | **0.02** | 23 | 28 | **31** | 33.8 | 33.5 | **32.3** |
| | 4 | 0.15 | 0.01 | **0.00** | 07 | 27 | **38** | **11.2** | 28.5 | 28.3 |
| | 6 | 0.01 | **0.00** | **0.00** | 10 | 36 | **39** | 12.4 | 19.2 | **11.5** |
| | 8 | 0.01 | 0.01 | **0.00** | 15 | 32 | **40** | 21.3 | 11.2 | **05.6** |
| 600 | 2 | 0.30 | **0.07** | **0.07** | 23 | 25 | **31** | 36.9 | **36.7** | 36.8 |
| | 4 | 0.13 | 0.02 | **0.00** | 09 | 15 | **36** | **09.7** | 22.1 | 42.2 |
| | 6 | 0.01 | **0.00** | **0.00** | 05 | 30 | **39** | **07.5** | 26.8 | 19.9 |
| | 8 | 0.01 | **0.00** | **0.00** | 06 | 33 | **40** | 16.0 | 22.0 | **15.1** |
| 800 | 2 | 0.27 | **0.04** | **0.04** | 22 | 27 | **31** | 41.0 | **30.0** | 34.4 |
| | 4 | 0.08 | **0.00** | 0.01 | 19 | 25 | **37** | **06.5** | 16.2 | 27.7 |
| | 6 | **0.00** | 0.01 | **0.00** | 09 | 24 | **38** | **06.2** | 32.6 | 35.2 |
| | 8 | 0.01 | **0.00** | **0.00** | 02 | 32 | **40** | **03.5** | 34.7 | 22.7 |
| 1000 | 2 | 0.16 | **0.05** | 0.06 | 20 | 28 | **32** | **30.8** | 35.0 | 40.0 |
| | 4 | 0.05 | **0.00** | **0.00** | 33 | 36 | **38** | **05.3** | 09.7 | 11.2 |
| | 6 | **0.00** | **0.00** | **0.00** | 11 | 17 | **37** | **11.3** | 19.8 | 46.2 |
| | 8 | **0.00** | **0.00** | **0.00** | 07 | 27 | **39** | **08.6** | 39.6 | 34.1 |
| Avg | | 0.10 | **0.01** | 0.02 | 16 | 29 | **37** | **17.1** | 24.3 | 24.9 |

**Table 9: Performance comparison of the proposed metaheuristics on the large set.**

| | | RPD (%) | | | Best count | | | Time to best ratio (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| N | M | LA | S1 | S2 | LA | S1 | S2 | LA | S1 | S2 |
| 40 | 2 | 0.33 | **0.02** | 0.10 | 08 | **12** | 10 | **21.4** | 24.4 | 29.3 |
| 60 | 2 | 0.66 | **0.03** | 0.12 | 07 | **13** | 09 | 33.9 | **19.9** | 36.5 |
| | 4 | 0.40 | **0.00** | 0.04 | 10 | **14** | 12 | **05.9** | 23.1 | 8.6 |
| 80 | 2 | 0.74 | **0.00** | 0.07 | 07 | **15** | 09 | **16.9** | 37.0 | 35.3 |
| | 4 | 0.52 | **0.00** | 0.10 | 12 | **15** | 12 | **05.6** | 18.9 | 19.9 |
| 100 | 2 | 0.67 | **0.01** | 0.10 | 07 | **12** | 06 | **09.6** | 48.5 | 40.8 |
| | 4 | 0.53 | **0.00** | 0.09 | 12 | **14** | 11 | **01.1** | 17.2 | 10.5 |
| | 6 | 0.73 | **0.00** | 0.12 | 12 | **15** | 12 | **04.0** | 16.3 | 12.1 |
| 120 | 2 | 0.62 | **0.02** | 0.07 | 08 | **13** | 08 | **11.0** | 41.4 | 53.6 |
| | 4 | 0.50 | **0.00** | 0.09 | 11 | **15** | 11 | **03.4** | 21.3 | 16.5 |
| | 6 | 0.70 | **0.00** | 0.23 | 12 | **15** | 12 | **04.9** | 16.2 | 18.2 |
| | 8 | 0.61 | **0.06** | 0.08 | 12 | **14** | 13 | **00.2** | 18.1 | 11.3 |
| Avg | | 0.58 | **0.01** | 0.10 | 10 | **14** | 10 | **09.8** | 25.2 | 24.4 |

**Table 10: Performance comparison of the proposed metaheuristics on the medium set.**

For the large set, LAHC is performing the worst out of the three metaheuristics with an overall RPD of 0.10% and approaching the other methods only on some classes. The two variants of simulated annealing have similar performance in terms of RPD. We notice that S2 performs better when the number of machines increases ($m \in \{6, 8\}$). This can be explained by the neighborhood operators of S2 which only focus on bottleneck machines hence showing the efficacy of such strategy. When $m$ is small (2 or 4), S1 will mostly perform moves on the bottleneck machines, giving it advantage compared to S2.

## 5 CONCLUSION

This work addressed the unrelated parallel machine scheduling with release dates and setup times. We proposed three heuristics: Best-Insertion-Based Approach (BIBA), Greedy Randomized Adaptive Search Procedure (GRASP) and Meta-heuristic for Randomized Priority Search (Meta-RaPS). Moreover, we introduced a Late-Acceptance Hill Climbing approach and two variants of Simulated Annealing. We derived five local search operators based on three moves: internal swap, external swap and external insertion. We introduced a three-set 1620-instance benchmark for the problem. The proposed benchmark is the first for the problem.

The proposed heuristics were compared and the results have shown that BIBA is the best heuristic among the three. The results also showed that BIBA yields the best performance when used to provide the initial solution for all the metaheuristics. When comparing the metaheuristics, no general trend can be derived. For the small set, it can be observed that LAHC performs best and yields the best RPD. For the medium sized instances, results show that the first variant of SA yields the best results. Last, for large instances, it is recommended that the second variant of SA using a restrictive strategy be used as it is performing best on the large set.

This study has shown that each metaheuristic performs best on each set. Possible future areas of research include proposing better heuristics and metaheuristics over all the benchmark. Moreover, it would interesting to derive lower bounds and investigate exact methods to evaluate the performance of the proposed approaches.

## REFERENCES

[1] Mojtaba Afzalirad and Javad Rezaeian. 2016. Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. *Computers & Industrial Engineering* 98 (2016), 40–52.

[2] Ibrahim M Al-harkan, Ammar A Qamhan, Ahmed Badwelan, Ali Alsamhan, and Lotfi Hidri. 2021. Modified Harmony Search Algorithm for Resource-Constrained Parallel Machine Scheduling Problem with Release Dates and Sequence-Dependent Setup Times. *Processes* 9, 4 (2021), 654.

[3] Ali Allahverdi. 2015. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research* 246, 2 (2015), 345–378.

[4] Jean-Paul Arnaout, Rami Musa, and Ghaith Rabadi. 2014. A two-stage Ant Colony optimization algorithm to minimize the makespan on unrelated parallel machines—part II: enhancements and experimentations. *Journal of Intelligent Manufacturing* 25, 1 (2014), 43–53.

[5] Jean-Paul Arnaout, Ghaith Rabadi, and Rami Musa. 2010. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing* 21, 6 (2010), 693–701.

[6] Oliver Avalos-Rosales, Francisco Angel-Bello, and Ada Alvarez. 2015. Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology* 76, 9-12 (2015), 1705–1718.

[7] Edmund K Burke and Yuri Bykov. 2017. The late acceptance hill-climbing heuristic. *European Journal of Operational Research* 258, 1 (2017), 70–78.

[8] Changyu Chen, Mahdi Fathi, Marzieh Khakifirooz, and Kan Wu. 2022. Hybrid Tabu Search Algorithm for Unrelated Parallel Machine Scheduling in Semiconductor Fabs with Setup Times, Job Release, and Expired Times. *Computers & Industrial Engineering* (2022), 107915.

[9] Gail W DePuy, Reinaldo J Moraga, and Gary E Whitehouse. 2005. Meta-RaPS: a simple and effective approach for solving the traveling salesman problem. *Transportation Research Part E: Logistics and Transportation Review* 41, 2 (2005), 115–130.

[10] Wei Fang, Haolin Zhu, and Yi Mei. 2022. Hybrid meta-heuristics for the unrelated parallel machine scheduling problem with setup times. *Knowledge-Based Systems* (2022), 108193.

[11] Luis Fanjul-Peyro. 2020. Models and an exact method for the Unrelated Parallel Machine scheduling problem with setups and resources. *Expert Systems with Applications: X* 5 (2020), 100022.

[12] Luis Fanjul-Peyro, Rubén Ruiz, and Federico Perea. 2019. Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *Computers & Operations Research* 101 (2019), 173–182.

[13] Paola Festa and Mauricio GC Resende. 2002. GRASP: An annotated bibliography. In *Essays and surveys in metaheuristics*. Springer, 325–367.

[14] Christian Gahm, Florian Denz, Martin Dirr, and Axel Tuma. 2016. Energy-efficient scheduling in manufacturing companies: A review and research framework. *European Journal of Operational Research* 248, 3 (2016), 744–757.

[15] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*. Vol. 5. Elsevier, 287–326.

[16] Xiaofeng Hu, Jin-Song Bao, and Ye Jin. 2010. Minimising makespan on parallel machines with precedence constraints and machine eligibility restrictions. *International Journal of Production Research* 48, 6 (2010), 1639–1651.

[17] Simin Huang, Linning Cai, and Xiaoyue Zhang. 2010. Parallel dedicated machine scheduling problem with sequence-dependent setups and a single server. *Computers & Industrial Engineering* 58, 1 (2010), 165–174.

[18] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. 2009. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36 (2009), 267–306.

[19] Kristijan Jaklinović, Marko Đurasević, and Domagoj Jakobović. 2021. Designing dispatching rules with genetic programming for the unrelated machines environment with constraints. *Expert Systems with Applications* 172 (2021), 114548.

[20] Jihene Kaabi and Youssef Harrath. 2014. A survey of parallel machine scheduling under availability constraints. *International Journal of Computer and Information Technology* 3, 2 (2014), 238–245.

[21] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. 1983. Optimization by simulated annealing. *science* 220, 4598 (1983), 671–680.

[22] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. 1977. Complexity of machine scheduling problems. In *Annals of discrete mathematics*. Vol. 1. Elsevier, 343–362.

[23] Yantong Li, Jean-François Côté, Leandro C Coelho, and Peng Wu. 2021. Novel efficient formulation and matheuristic for large-sized unrelated parallel machine scheduling with release dates. *International Journal of Production Research* (2021), 1–20.

[24] Maximilian Moser, Nysret Musliu, Andrea Schaerf, and Felix Winter. 2021. Exact and metaheuristic approaches for unrelated parallel machine scheduling. *Journal of Scheduling* (2021), 1–28.

[25] Michele Pfund, John W Fowler, and Jatinder ND Gupta. 2004. A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling problems. *Journal of the Chinese Institute of Industrial Engineers* 21, 3 (2004), 230–241.

[26] Michael Pinedo. 2012. *Scheduling*. Vol. 29. Springer.

[27] Mourad Terzi, Taha Arbaoui, Farouk Yalaoui, and Karima Benatchba. 2020. Solving the unrelated parallel machine scheduling problem with setups using late acceptance hill climbing. In *Asian Conference on Intelligent Information and Database Systems*. Springer, 249–258.

[28] Marko Đurasević and Domagoj Jakobović. 2021. Heuristic and Metaheuristic Methods for the Unrelated Machines Scheduling Problem: A Survey. *arXiv preprint arXiv:2107.13106* (2021).

[29] Eva Vallada and Rubén Ruiz. 2011. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research* 211, 3 (2011), 612–622.

[30] Mario C Vélez-Gallego, Jairo Maya, and Jairo R Montoya-Torres. 2016. A beam search heuristic for scheduling a single machine with release dates and sequence dependent setup times to minimize the makespan. *Computers & Operations Research* 73 (2016), 132–140.

[31] I-Lin Wang, Yi-Chi Wang, and Chih-Wei Chen. 2013. Scheduling unrelated parallel machines in semiconductor manufacturing by problem reduction and local search heuristics. *Flexible Services and Manufacturing Journal* 25, 3 (2013), 343–366.

[32] Kuo-Ching Ying, Zne-Jung Lee, and Shih-Wei Lin. 2012. Makespan minimization for scheduling unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing* 23, 5 (2012), 1795–1803.