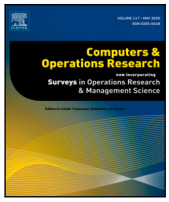
Computers & Operations Research 139 (2022) 105637

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



Application of a hybrid evolutionary algorithm to resource-constrained parallel machine scheduling with setup times

Leonardo C.R. Soares [a,b,*], Marco A.M. Carvalho [b]

[a] Instituto Federal do Sudeste de Minas Gerais - Campus Manhuaçu, Manhuaçu, Minas Gerais 36909-300, Brazil
[b] Universidade Federal de Ouro Preto - Campus Morro do Cruzeiro, Ouro Preto, Minas Gerais 35400-000, Brazil

ARTICLE INFO

Keywords:
Combinatorial problems
Metaheuristic
Biased random-key genetic algorithm
Resource-constrained parallel machine scheduling with setup times

ABSTRACT

We address the problem of resource-constrained parallel machine scheduling with setup times in the practical context of microelectronic components manufacturing. This $\mathcal{NP}$-hard problem is addressed using a biased random-key genetic algorithm hybridized with tailored local search procedures organized using variable neighborhood descent. The only benchmark available in the literature is utilized, and the optimal results are presented for all instances. Two new sets with 270 challenging instances are proposed to assess the quality of the solutions reported by the proposed method. A series of experiments are conducted to generate lower and upper bounds, including four models, two list processing heuristics from the literature and an implementation of a general variable neighborhood search metaheuristic. The bounds are used as reference values. The average percentage distances from the lower and upper bounds were 22.44% and −7.62%, respectively.

1. Introduction

A typical flexible manufacturing system (FMS) comprises a network of flexible machines connected by an automatic material handling system. The main feature of such flexible machines is their ability to manufacture different products based on different machine setups. Each step of product manufacturing is referred to as a *job*, which is processed by a flexible machine and may require a setup of a series of *resources* such as tools, pallets, and molds. The job sequencing and tool switching problem (SSP) is a well-known FMS single-machine job scheduling problem in which jobs require a set of tools to be installed in the magazine of a flexible machine before the processing starts. Between two consecutively scheduled jobs, tool switches occur in the magazine to generate a proper setup for the job processing, implying the addition of extra setup time to the machine completion time. In addition to the job scheduling, a rational plan for tool switching is required to maximize the productivity of the FMS.

The identical parallel machines problem with tooling constraints (IPMTC) (Soares and Carvalho, 2020) is a generalization of the SSP that considers more than one machine in the environment. Decisions concern the assignment of jobs to machines, the sequencing of jobs assigned to each machine, and the plan for tool switching in each machine. Herein, the tools are not shared, that is, each machine has its own complete set of tools. A special case of the IPMTC, in which shared resources restrict a job process in an environment comprising flexible parallel machines is called resource-constrained parallel machine scheduling (RCPMS). First addressed by Garey and Graham (1975), the RCPMS is an $\mathcal{NP}$-hard problem because each machine in this problem is an instance of the SSP, which is a proven $\mathcal{NP}$-hard problem (Crama et al., 1994).

Among the several examples of possible resource constraints in SMFs, we mention (i) the unavailability of raw material at some point in the production (Pinheiro and Arroyo, 2020), (ii) unavailability of machines either owing to the characteristics intrinsic to jobs that require processing on specific machines (Li et al., 2020) or to maintenance stops (Reddy et al., 2018), (iii) and the sharing of tools by flexible machines (Keung et al., 2001) including environments in which the flexible machines do not have magazines for storing the tools (Agnetis et al., 1997). These limitations can occur either in combination or isolation.

A practical application of the RCPMS is seen in the industries that manufacture microelectronic components (Chung et al., 2019), especially microchips. Typically, hundreds of circuits are printed on a semiconductor wafer using a process named photolithography. The layout of each circuit is created in a pattern tool known as mask, which is used as a mold during the circuit printing process and must be assembled in a flexible machine before a specific microchip type is produced. The operation of switching molds to produce different microchip types in a flexible machine adds setup time to the job-scheduling problem. The machines and molds used in this process have

* Corresponding author at: Instituto Federal do Sudeste de Minas Gerais - Campus Manhuaçu, Manhuaçu, Minas Gerais 36909-300, Brazil.
E-mail addresses: leonardo.soares@ifsudestemg.edu.br (L.C.R. Soares), mamc@ufop.edu.br (M.A.M. Carvalho).

https://doi.org/10.1016/j.cor.2021.105637
Received 13 May 2021; Received in revised form 22 September 2021; Accepted 10 November 2021
Available online 24 November 2021

high costs (Chung et al., 2019), and considering that every machine has the ability to produce any model of a microchip, it is usual for the number of copies of each mold to be less than the number of machines, thereby implying sharing of molds between machines.

Whenever a flexible machine requires a mold type that is used by another machine, a *conflict* occurs and causes idle time owing to the unavailability of the mold; the idle time may continue until the mold is released. In large-scale production environments with parallel machines, the use of molds must be planned so that production is optimized, by minimizing the machine idle time caused by mold conflicts or extra setup times caused by unnecessary mold switches.

This study is motivated by the practical application of RCPMS in the aforementioned context. The objective is to minimize the makespan, that is, the completion time of the *critical* machine (the most time-consuming machine), which is achieved by assigning jobs to machines in such a manner that the idle and setup times are minimized. As the main contribution, this paper first presents the development of a biased random-key genetic algorithm (BRKGA) hybridized with RCPMS-tailored local search procedures organized using variable neighborhood descent (VND) to approach the RCPMS. Second, we publish the optimal results for all instances of the single set of benchmark instances available in the literature and propose a new benchmark comprising more difficult problems, to build up on the previous research and consistently contribute to the future studies of the problem.

The remainder of this paper is organized as follows. Section 2 formally describes the RCPMS. Section 3 reviews the existing literature on RCPMS. Section 4 presents the details of the proposed BRKGA metaheuristic and its hybridization. The computational results are presented in Section 5. Finally, the conclusions and future research directions are provided in Section 6.

2. Problem description

To define the RCPMS, consider an FMS comprising a set of identical parallel machines, $M = \{1, \dots, m\}$, where each machine has the capacity to handle exactly one mold at a time; a set of molds, $T = \{1, \dots, l\}$, containing exactly one copy of each mold type; a set of jobs, $J = \{1, \dots, n\}$, wherein each job has a processing time indicated by p_j $(j \in J)$ and a required mold indicated by t_j $(j \in J)$; and finally, a constant setup time, $\bar{p}$, which is required to perform each switch of the molds in a machine. The RCPMS involves assigning jobs to machines and determining their processing sequence such that the makespan, C_{max}, is minimized. Additionally, the jobs have no release or due time and are not preemptive, and there are no precedence constraints. Only one job can be processed at a time by a machine, and the mold required by the job must be assembled in the flexible machine before the job is processed. Two or more jobs requiring the same mold cannot be processed simultaneously on different machines. Each machine is eligible for every mold assembly, and the setup time required for the initial machine configuration is not considered in the makespan calculation.

Formalization of this practical application of RCPMS is very similar to the definition of IPMTC, which is also a generalization of the SSP. The tools are analogous to the molds; therefore, the RCPMS can be considered to be a special case of IPMTC where (i) each job requires exactly one tool, (ii) the set of all tools is shared among machines, and (iii) the capacity of the magazine is always equal to one. According to the notation provided by Graham et al. (1979), the application of RCPMS can be denoted as $P_m|s_{jk}, res1|C_{max}$, where P_m indicates an environment with identical parallel machines, s_{jk} refers to the sequence-dependent setup times, $res1$ refers to the resource constraint, and C_{max} denotes the objective function of minimizing the makespan.

Solution for an RCPMS instance is given by the sequencing of jobs on each machine and indicates the intervals of idle time. Owing to the resource constraint, whenever the requested mold for job processing is in use by another machine, an idle interval is generated until its release. The mold switches on a machine, $i \in M$, as in the SSP and IPMTC, can be represented by a binary matrix, $R^i = \{r^i_{tj}\}$, with $t \in T$ and $j \in J$. The rows of R^i represent the molds, and the columns represent the jobs scheduled in machine i in the order of processing. An element $r^i_{tj} = 1$ indicates that mold t is assembled on machine i during the processing of job j with $r^i_{tj} = 0$ otherwise.

The number of inversions from 0 to 1 of matrix R^i indicates the number of mold switches, and it can be calculated by Eq. (1), as proposed by Crama et al. (1994) for the SSP.

$$Z_{SSP}(R^i) = \sum_{j \in J} \sum_{t \in T} r^i_{tj}(1 - r^i_{tj-1}). \tag{1}$$

The completion time of a machine is given by the sum of the job processing times, machine idle time, and setup time for mold switches. Let I_j be the idle time generated immediately before the processing of job j, and let $x_{ij} = 1$ indicate that job j is assigned to machine i ($x_{ij} = 0$, otherwise). As shown in Eq. (2), the RCPMS objective function minimizes the makespan.

$$C_{max} = \max\left\{ \sum_{j \in J} x_{ij} p_j + \sum_{j \in J} x_{ij} I_j + Z_{SSP}(R^i) \times \bar{p} \right\}, \ \forall i \in M \tag{2}$$

3. Related work

Owing to the wide practical application of the RCPMS, related studies since the 1970s can be found in the literature. First introduced by Garey and Graham (1975), the inclusion of resource constraints in a machine was described by the authors as a mechanism to offer greater realism to the scheduling models. Over the years, scheduling problems with resource constraints have been addressed in several areas, such as nurse scheduling (Ohki et al., 2006), sports competitions (Guangdong et al., 2007), and agro-food industries (Karray et al., 2011), and others (Pinedo, 2008).

Specific literature on RCPMS with mold constraints is limited, and most of the works are case studies. In these studies, the results obtained were compared with manual scheduling or with other methods proposed in the same study. The main practical applications of RCPMS with mold constraints include the molding of plastic or metal products and the production of electronic components.

Tamaki et al. (1993) addressed job scheduling on unrelated parallel machines with mold constraints. The problem originates in a plastics-forming plant where several types of pipes are manufactured. Each pipe requires a specific type of mold for its manufacturing. The molds are not universal; therefore, compatibility between the machine and the mold is required. The quantity of each mold is limited, and the setup times for the assembly and disassembly of molds were considered. In addition to providing a mathematical formulation for the problem, the authors presented an implementation of the simulated annealing metaheuristic, a genetic algorithm, and two methods based on local search. Three metrics of evaluation were considered in the computational experiments: makespan, average lateness, and maximum lateness. The implemented methods were compared with each other using two instances that were proposed in the same paper. The results reported by the simulated annealing metaheuristic outperformed the other implementations.

Gao et al. (1998) presented a production scheduling system for an electrical appliance company. This company produces plastic components for home appliances using parallel machines. Each component has a certain processing time and uses a specific mold. There is a compatibility relationship between the machines and molds, and the number of copies of each mold is limited. Only the setup time for mold assembly was considered. A mathematical model and a heuristic based on the list processing were presented. The approaches were evaluated using a real instance and were compared to manual scheduling, and the results reported an average reduction of 68.00% in the daily processing time.

The abovementioned environment was later addressed by Xianzhang and Chengyao (1999). Aiming at minimizing the total tardiness, the

authors presented a heuristic based on list processing. The computational experiments considered a large real-world instance. The obtained results when compared with those of manual scheduling achieved a 93.31% reduction in total tardiness.

The production of plastic pipes was again considered in the study by Chen (2005) with the objective of minimizing the makespan. Herein, an implementation of the tabu search metaheuristic was presented, and 145 artificial instances proposed in the study were considered in the experiments. The reported results were compared with an implementation of simulated annealing and with the optimal results generated by a mathematical model, both proposed by Tamaki et al. (1993). The tabu search reported the best results, including the optimal results for 24 small instances.

Considering the same environment, but with the objective of minimizing the total tardiness, Chen and Wu (2006) presented a hybrid heuristic that combines the threshold accepting method and tabu search. A total of two sets of instances proposed in the study were considered in the experiments. For the first set, which comprises the smallest problems, the proposed method reported optimal solutions for all instances; however, the method used to generate the optimal results is not mentioned. For the second set, which comprises the largest problems, the obtained values were compared with three other methods: the simulated annealing proposed by Tamaki et al. (1993), a standard version of the threshold accepting method, and an adaptation of the apparent tardiness cost with setups rule that was extended by Lee and Pinedo (1997) to consider sequence-dependent setup times. The proposed hybrid heuristic reported the best solution values among the compared methods.

In an environment with unrelated parallel machines, Hong et al. addressed the scheduling of jobs that can be processed using any mold from a subset of eligible molds. Each mold can be assembled on a subset of compatible machines. The processing speeds of the jobs vary according to the machine. The problem was modeled mathematically and solved using a heuristic based on list processing. The experiments used real data from the steel pipe industry. The total tardiness, number of tardy jobs, and average tardiness obtained by the heuristic were compared with the respective values obtained by manual scheduling. The authors reported that the proposed approach significantly improved the results, although specific values were not reported.

Scheduling of jobs with mold constraints on identical parallel machines was first addressed by Hong et al. (2009). In this environment, there was only one copy of each mold, and the setup time was considered. To minimize the makespan, two versions of a genetic algorithm were proposed. The first version simply discards infeasible solutions, and the second employs an adjustment operator that eliminates conflicts by reallocating conflicting jobs. Computational experiments compared the two genetic algorithms and concluded that the adjustment operator effectively improved the reported solutions.

The study by Xiong et al. (2013) considered the scheduling of the batches of parts that must be molded and painted, considering mold constraints. All parts of the same batch use the same subset of molds and must be painted in the same color. Each mold can be assembled on a subset of machines, and the processing speed of each machine differs depending on the mold assembled on it. In addition to the setup time for assembling and disassembling molds, changing the colors between two sequential batches may require extra setup time. To minimize the weighted total tardiness, the authors presented a genetic algorithm and two heuristics based on list processing. Only one artificial instance was used in the experiments. The experiments revealed that the genetic algorithm outperformed the heuristics.

Bitar et al. (2016) proposed a memetic algorithm to solve the unrelated parallel machine scheduling with additional resources in a semiconductor plant. Each job requires a mold from a subset of available molds for its processing. The subset of molds is shared between machines. There is only one copy of each mold type and the setup time for mold assembly is considered. Two objective functions were considered, the maximization of the number of produced wafers and the minimization of the weighted completion times. The computational experiments used a set of instances proposed in the work itself and aimed at determining the best configurations for the proposed algorithm. The results were not compared with other works or reference values.

Lee et al. (2014) presented a genetic algorithm hybridized with local search to minimize the makespan on unrelated parallel machines. Each job requires a mold from a subset of available molds for its processing. Each mold can be assembled on a subset of compatible machines. A total of nine artificial instances proposed in the study were considered in the experiments. Two versions of a genetic algorithm were compared: one standard and the other, hybridized with a local search. The authors concluded that the incorporation of local search into the genetic algorithm effectively contributed to improving the quality of the solutions obtained; however, the reported results are not sufficiently detailed.

Chung et al. (2019) approached job scheduling on two identical parallel machines with mold constraints. They used only one copy of each mold and did not consider the setup time. With the objective of minimizing the makespan, a mathematical model and two list processing heuristics were presented. To assess the quality of the proposed heuristics, the authors additionally presented implementations of the discrete particle swarm optimization (DPSO) and variable neighborhood search (VNS) metaheuristics. Both these implementations were standard, that is, no particular characteristics of the problem were considered. The results reported by the heuristics were evaluated with respect to a lower bound given by the solution of a relaxed version of the mixed-integer linear programming formulation by Mokotoff (2004). The relaxed version of the formulation disregards the RCPMS resource constraints.

Two sets of instances proposed in this study are considered in the experiments. Optimal results were generated for all instances of the first set by the proposed mathematical model, whereas the implementations of the VNS and DPSO metaheuristics reported average percentage distances of 0.002% and 0.010%, respectively. Both the list processing heuristics were applied to each instance of this set, and the combination of their best results generated an average percentage distance of 1.270%. The second set contained the highest number of problems. The proposed model proved to be ineffective, considering the 1200 s running time limit. In comparison to the reported lower bounds, the combined list processing heuristics generated the best results, reporting an average percentage distance of 0.18%. The VNS and DPSO reported average percentage distances of 0.98% and 1.42%, respectively.

Specifically for RCPMS in the context of mold constraints, the instances of Chung et al. (2019) are the only benchmarks available in the literature. However, the environment of the two machines and no setup times have restricted application. Additionally, the short percentage distances between the list processing heuristics and lower bound values, obtained by relaxing the resource constraint indicate that such benchmarks mostly comprise trivial instances, in which, the use of molds has no substantial impact on the difficulty of problems. Therefore, there is a gap in benchmarking in the RCPMS literature, thereby requiring new instances that represent both a greater challenge and broader application for the proposed methods, thereby enabling the progression of the state-of-the-art related to the problem. In the computational experiments reported in Section 5, the existing benchmark was analyzed in detail and a new benchmark was proposed. Both benchmarks were considered to assess the quality of the method proposed in the study.

The reader interested in general FMS operations, models, and related problems, including the SSP and IPMTC, can refer to the comprehensive and recent surveys of Allahverdi (2015) and Calmels (2018).

1.575	2.884	1.024	2.198	2.276	2.754	2.058	1.038

Fig. 1. Encoding of an individual for an instance of RCPMS with two machines and eight jobs.

4. Methods

This section describes the proposed implementation of a BRKGA hybridized with a VND procedure. The BRKGA was selected owing to its suitability for permutation problems and its recent strong performance on relevant combinatorial problems (Soares and Carvalho, 2020; A. F. Kummer et al.; Andrade et al., 2017; Ramos et al., 2018; Oliveira et al., 2019). Hybridization was proposed with the purpose of introducing problem-specific information into the evolutionary process, thereby strengthening the intensification of the search by performing a systemic local search.

4.1. Biased random-key genetic algorithm

Random-key genetic algorithm (RKGA), which is a variant of the well-known genetic algorithm, was introduced by Bean (1994). In this variant, *individuals* have their *genes* (i.e., a part of their information) encoded as vectors of randomly generated real numbers (or keys) in the continuous interval [0, 1]. Similar to the standard genetic algorithm, an initial *population* of pop randomly generated individuals evolves over a number of *generations* through the application of selection, reproduction, and mutation operators. Each individual represents a potential solution for the problem at hand, and its evaluation produces a *fitness* value. The pop_e individuals with the best fitness values are considered the *elite* of the population.

Through the evolutionary process, the population of the next generation is created based on the population of the previous generation. The RKGA copies the elite individuals of one generation to the next. Additional pop_m individuals are generated randomly as *mutants* and introduced into the new population. The remaining portion of the new population comprises offspring individuals, which are generated by combining two *parent* individuals selected randomly from the previous population and using the *parameterized uniform crossover* (Spears and De Jong, 1995).

Motivated by the RKGA, Gonçalves et al. (2012) introduced the BRKGA. In this genetic algorithm, one parent is always chosen randomly from the elite group in the reproduction operator. Additionally, although the parameterized uniform crossover is again employed, the offspring are more inclined (i.e., biased) to inherit the genes of the parent individual belonging to the elite group. Although the difference between the two algorithms is small, according to Gonçalves, Resende, and Gonçalves et al. (2012), the BRKGA almost always generates better solutions than the RKGA.

4.1.1. Encoding and decoding

Similar to that in RKGA, each individual in BRKGA is encoded using a vector of x random keys in the continuous interval $[0, 1] \in \mathbb{R}$. To accurately represent an RCPMS solution, the encoding must contain the information concerning the assignment of jobs to machines and their sequencing. Therefore, we changed the key value range to $[1, m+1) \in \mathbb{R}$, where m is the total number of machines in an instance. Fig. 1 illustrates the encoding of an individual for an RCPMS instance comprising two machines and eight jobs.

In the decoding step, the keys are sorted in an non-decreasing value order, maintaining their original index in the vector. The integer part, i, of each key denotes the machine index to which job t_j, identified by the original index j in the vector, is assigned to. The scheduling of a job is determined by its position among the sorted keys. The individual from Fig. 1 is sorted as illustrated in Fig. 2. This example corresponds to the assignment of jobs 2, 7, and 0 to machine 1 and jobs 6, 3, 4, 5, and 1 to machine 2, in the given order.

Fitness of an individual is the makespan of the solution, and it is calculated according to Eq. (2) after the decoding step.

4.1.2. Hybridization

To effectively explore the search space and reduce the BRKGA decoupling from the RCPMS, we propose a hybridization with four tailored local search procedures that consider characteristics inherent to the RCPMS. The proposed hybridization is performed in the decoding step as the second component of search intensification. This step is particularly suitable for parallelization and has the potential to significantly reduce the running times of the hybrid BRKGA.

Algorithm 1 presents the pseudocode for the proposed hybrid BRKGA. The input comprises the number of jobs (n), number of machines (m), population size (pop), elite group size (pop_e), number of mutant individuals (pop_m), and the stopping criterion ($stopping\_criterion$). The initial population is randomly generated (line 1). The main loop (lines 2 to 22) evolves the population until the stopping criterion is met. Each new individual is decoded into a solution into the RCPMS and evaluated (lines 3 and 4). The VND is performed for all individuals in the elite group (line 5), whereas the local search is performed exactly once (line 6) for the non-elite individuals. All the decoded solutions are encoded back as individuals to reflect possible changes, and their fitness is updated (line 7). Subsequently, a new elite group is formed (lines 8 and 9). The new elite group and mutant individuals are inserted into the population of the next generation (lines 10 and 11). The internal loop (lines 12 to 22) generates the offspring individuals. Two parents are randomly selected, one from the elite group (line 13) and one from the non-elite group (line 14). Next, the offspring individual inherits each of its genes (lines 15 to 21) either from the elite parent (line 18) with probability ρ (line 16) or from the non-elite parent (line 20). Each new individual is added to the next generation (line 21). Upon completion, the next population succeeds the previous one (line 22). The solution to an RCPMS instance is obtained by decoding the individual with the lowest fitness value at the end of the evolutionary process. Each BRKGA and VND component is discussed in detail in the following sections.

4.2. Variable neighborhood descent

Introduced by Mladenović and Hansen (1997), VND is based on the regular change in the neighborhood structure during the exploration of the search space. Given an initial solution, the neighborhoods are organized and explored sequentially. Whenever an improved neighbor solution is found, the exploration is restarted from the first neighborhood. If there is no neighbor solution with a better solution value, the next neighborhood is explored. The algorithm ends after exploring the last neighborhood without finding an improved solution, that is, when a local optimum shared by all neighborhoods is found.

The steps of our VND implementation are described in Algorithm 2. Given a feasible solution s as the input, the sequence of neighborhoods is defined by a control variable (line 1). The main loop (lines 2 to 15) coordinates the exploration of neighborhoods until a local optimum common to all neighborhoods is found. Then, the neighborhoods are explored sequentially. First, the job insertion (lines 3 and 4) neighborhood is explored, followed by job exchange (lines 5 and 6), job relocation (lines 7 and 8), and 1-block grouping (lines 9 and 10). Whenever a neighboring solution with the best solution value is found, the current solution, s, is updated and the exploration is restarted by the first neighborhood (lines 11 to 13). If there is no solution improvement in the current neighborhood, the next neighborhood is selected (lines 14 and 15). When the exploration of all the neighborhoods does not result in solution improvement, the algorithm ends and the current solution is returned (line 16).

Original Index	2	7	0	6	3	4	5	1
Keys	1.024	1.038	1.575	2.058	2.198	2.276	2.754	2.884

Fig. 2. Sorted encoding of an individual during the decoding process.

Algorithm 1: Hybrid biased random-key genetic algorithm

```
input : n, m, pop, pop_e, pop_m, stop_criterion
Generate population Π with pop vectors of n random keys
  chosen from [1, m + 1);
while stop_criterion is not satisfied do
    Decode each new individual;
    Evaluate the fitness for each decoded individual in Π;
    Apply the VND on the pop_e first individuals;
    Apply each local search procedure once on the remaining
      individuals;
    Encode each solution as individuals and update their
      fitness;
    Sort individuals in the order of non-decreasing fitness s;
    Group the first pop_e individuals in Π_e;
    Initialize the population of the next generation, Π+ ← Π_e;
    Generate a set of pop_m mutants and add it to Π+;
    for i ← 1 to pop−pop_e−pop_m do
        Select random individual a from the elite group;
        Select random individual b from the non-elite group;
        for j ← 1 to n do
            Throw a biased coin with probability ρ of heads;
            if heads then
                c[j] ← a[j];
            else
                c[j] ← b[j];
        Π+ ← Π+ ∪ {c};
    Π ← Π+;
```

Algorithm 2: Variable neighborhood descent

```
input : solution s
k = 1;
while k ≠ 5 do
    if k = 1 then
        s′ ← Job Insertion(s);
    if k = 2 then
        s′ ← Job Exchange(s);
    if k = 3 then
        s′ ← Job Relocation(s);
    if k = 4 then
        s′ ← 1-block Grouping(s);
    if C_max (s′) < C_max (s) then
        s ← s′;
        k ← 1;
    else
        k ← k + 1;
return s;
```

Our VND implementation includes four neighborhoods, and their sequence was defined according to the characteristics of each, with an aim to maximize their effect on search intensification. Initially, to correct potentially unbalanced solutions, the job insertion is applied, which tends to produce a greater balance in the job distribution between the machines. After that, when no more space for simple insertions remains, the job exchange local search attempts cross insertions between the machines. Furthermore, job relocation attempts to eliminate idle intervals on the critical machine (machine with the longest processing time) by reallocating jobs that can occupy that interval. This reduces the machine processing time and eventually creates space for new insertions. Finally, the 1-block grouping attempts to concentrate jobs that require the same mold by reducing the number of mold switches and releasing the mold to avoid conflicts. Each neighborhood is described in detail and illustrated next. Section 5 discusses their effectiveness and contributions to the performance of hybrid BRKGA.

As aforementioned, the BRKGA copies the individuals belonging to the elite group from one generation to the next. Knowing that the application of the VND results in a local optimum, elite individuals are marked to avoid a second and unnecessary application of local search in future generations.

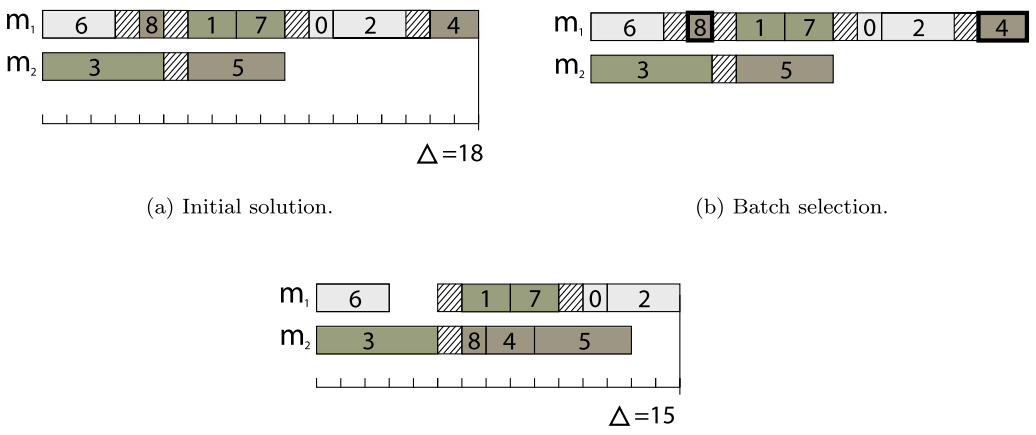
4.3. Job insertion

To maximize the productivity of the machines, and consequently, to minimize the makespan, the job insertion local search attempts to balance the processing times among the machines. To this end, jobs assigned to the critical machine are analyzed in batches defined by each mold type. Next, the shortest processing time batch is removed and inserted into the machine with the shortest processing time. Although these jobs may not be scheduled sequentially on the critical machine, they are included contiguously in the new machine. The batch is inserted immediately before the first job that is already assigned to the new machine and which requires the same mold as that of the batch. If there are no such jobs, the batch is inserted at the end of the schedule. The solution is evaluated at each move, and if the solution value improves, the local search is restarted considering the current critical machine. The complexity of this move is bounded by $\mathcal{O}(n)$. However, given the possibility of the move causing idle intervals on different machines owing to conflicts, the entire solution needs to be reevaluated, with complexity bounded by $\mathcal{O}(n^2)$, which determines the complexity of this local search.

Fig. 3 illustrates a hypothetical move of this local search. Jobs are represented by rectangles whose widths are proportional to the processing times. The mold required for each job is identified by the color of the rectangle and the setup time for the mold switch is represented by the hatched rectangle of proportional width. The same pattern applies to the other illustrations. A feasible solution comprising two machines is presented in (a). Machine m_1 has the longest processing time; therefore, its jobs are analyzed in batches. The selection of the batch with the shortest processing time, which comprises jobs [8, 4] is highlighted in (b). Furthermore, as shown in (c), the batch is removed from machine m_1 and inserted in machine m_2, immediately before job 5. The resulting neighbor solution is evaluated, and it shows improvement over the previous solution value.

4.4. Job exchange

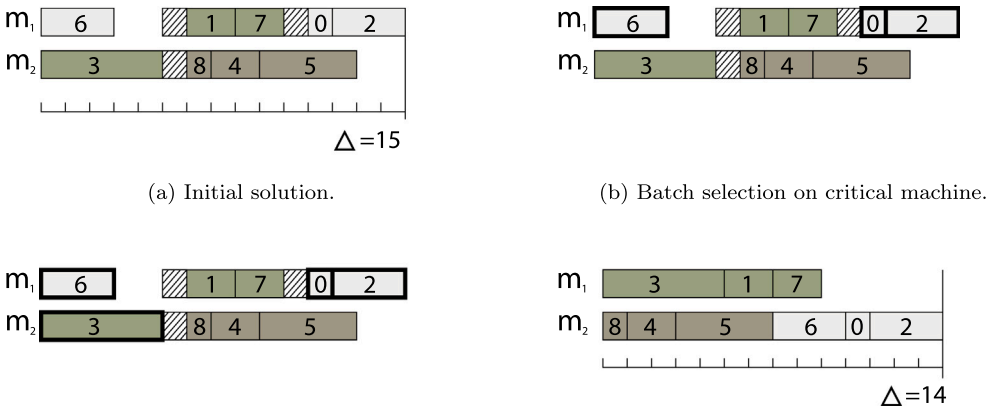
Exchanging jobs between the critical machine and the machine with the shortest processing time can reduce the makespan if the processing times of the jobs are different or if the exchange results in shorter setup times. Following this rationale, this local search conducts analysis in batches defined by each mold type and exchanges the longest processing time batch assigned to the critical machine with the shortest processing time batch assigned to the machine with the shortest processing time. Moreover, the batches are inserted immediately before



(a) Initial solution. (b) Batch selection.

(c) Neighbor solution.

Fig. 3. Job insertion local search.



(a) Initial solution. (b) Batch selection on critical machine.

(c) Batch selection on non-critical machine. (d) Neighbor solution.

Fig. 4. Job exchange local search.

the first job that is already assigned to the new machine, and which requires the same mold as that of the batch. If there are no such jobs, the batches are inserted at the end of the schedule. The solution is evaluated at each move, and the local search ends when no move is able to improve the solution. The exchange move has complexity bounded by $\mathcal{O}(n)$, but the complexity is determined by the evaluation function, $\mathcal{O}(n^2)$.

A hypothetical move performed by this local search is illustrated in Fig. 4. A feasible solution comprising two machines is presented in (a). Machine m_1 is the critical machine, and the longest processing time batch [6, 0, 2] is highlighted in (b). In (c), the shortest processing time batch [3] on the machine with the shortest processing time is highlighted. The batches are then exchanged. On machine $m1$, job 1 requires the same mold as the batch being inserted; therefore, the batch is inserted immediately before this job. On machine $m2$, the mold required by the batch is not used by any other job; therefore, the batch is inserted at the end of the schedule. After the local search move, the critical machine changes. The resulting neighbor solution with an improved solution value is presented in (d).

4.5. Job relocation

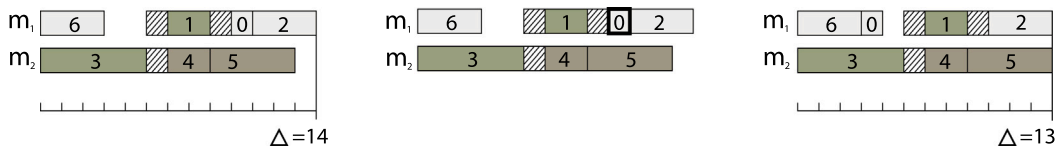
The idle intervals caused by mold conflicts in the critical machine directly affect the solution value. The job relocation local search aims to fill these gaps in machine productivity. The machines are sorted by non-increasing completion times. All idle intervals on the critical machine are identified, and then, the procedure searches for jobs not requiring the conflicting mold and, that the combined setup and processing times are shorter or equal to the duration of the idle interval. If there is more than one candidate job, that job with the combined setup and processing time closest to the duration of the idle interval is selected. The selected job is placed immediately after the job that precedes the idle interval. The resulting solution is evaluated, and if there is an improvement in its value, the process restarts.

As a side effect, a move may create new mold conflicts and idle intervals. If this occurs on the critical machine, the idle intervals are handled by the procedure restart. Otherwise, the idle intervals do not directly affect the solution value; therefore, they are ignored. The local search ends when there are no moves that result in solution improvement. Similar to the previous local search procedures, although the complexity of this move is bounded by $\mathcal{O}(n)$, the overall complexity is bounded by the $\mathcal{O}(n^2)$ evaluation procedure.

Fig. 5 illustrates a hypothetical move of this local search. A feasible solution comprising two machines, wherein m_1 has the longest processing time and a single idle interval is presented in (a). In (b), the only candidate job is highlighted. Job 0 is moved to the beginning of the idle interval and the resulting improved neighbor solution is presented in (c).

4.6. 1-Block grouping

In binary matrices, a consecutive sequence of '1' elements in the same row is known as a *1-block*. The '0' elements that separate such



(a) Initial solution. (b) Job selection. (c) Neighbor solution.

Fig. 5. Job relocation local search.

1-blocks are known as *discontinuities*. Based on such concepts, Paiva and Carvalho (2017) in the context of the SSP, presented a 1-block grouping local search to reduce the number of tool switches in a flexible machine. The rationale of this local search is to prevent the removal of frequently required tools. The same rationale can be applied to binary matrix R^{i^*} of the mold switches (see Section 1), where i^* denotes the critical machine. Therefore, minimizing the number of mold switches may lead to improved solutions. Even in the case of $\bar{p} = 0$, grouping jobs with the same mold can lead to an improved solution by increasing the number of available molds in a given time interval.

Given an RCPMS solution, matrix R^{i^*} is analyzed row-wise in a random order, to search for discontinuities. In case of a discontinuity, the local search moves the columns of the 1-block that precedes the discontinuity to the left and to the right of the 1-block that succeeds the discontinuity. At each move, the solution is evaluated, and the move that results in the best solution value is performed. In the event of a tie, the previous move is maintained. Only solution-improving moves are performed, and the local search ends when no move is able to improve the solution value. Each 1-blocks grouping move has complexity bounded by $\mathcal{O}(n)$. At each one of the n possible moves, the solution is evaluated in time $\mathcal{O}(n^2)$; therefore, the complexity of this local search is bounded by $\mathcal{O}(n^3)$.

Fig. 6 illustrates a hypothetical move of this local search. In (a), matrix R^{i^*} of a feasible solution and its makespan are presented. In (b), row 2 is randomly selected for analysis—it has two 1-blocks and one discontinuity. The local search moves the columns of the first 1-block to the left and to the right of the second 1-block. As shown in (c), the first move attempt improves the solution value, whereas the second (d) move attempt results in a worse solution, and therefore, it is discarded. The resulting improved neighbor solution is presented in (c).

5. Computational experiments

A set of computational experiments was performed to assess the quality of the proposed hybrid BRKGA and analyze its performance. The following sections describe the instances considered and discuss the experimental results. The detailed results of these experiments are provided in the Supplementary Material. The computational environment of the computational experiments included a computer with Intel Xeon E5-2660 2.2 GHz processor, 384 GB RAM, and CentOS 6.8 operating system. The hybrid BRKGA was implemented in C++ and compiled using GCC 10.1.0 with options -O3 and -march = native. The OpenMP (OpenMP, 2019) shared-memory parallel programming API version 5.0 was used to parallelize the decoding step.

5.1. Instances

Specifically, for mold constraints, the RCPMS has only two sets of benchmark instances available in the literature, and both were proposed by Chung et al. (2019). The first set of small problems comprises 260 instances with $n \in \{6, 9, 10, 15\}$. The second set of large problems comprises 320 instances with $n \in \{20, 50, 100, 150\}$. For both these sets, $l \in \{4, 6, 8, 10\}$, $m = 2$, and the setup times are not considered. However, the low percentage distance values reported by the list processing algorithms regarding the solution of a relaxed version of the problem indicate that such instances are not only trivial but also represent a restricted scenario. In fact, as reported in Section 5.6, the solution values generated by the hybrid BRKGA are equal to a trivial lower bound for all those instances. Therefore, for a precise evaluation of the proposed method and consistent progression in the study of the RCPMS, two new sets of instances were introduced.

Beezão et al. (2017) while addressing the IPMTC, presented two sets of benchmark instances. These instances meet the requirements of a large number of machines and different setup times. The RCPMS is a special case of the IPMTC; therefore, we decided to adapt such benchmarks for the RCMPS. The original number of machines, jobs, and setup times were maintained. The job processing times are generated using a uniform distribution over the interval [1,59]. The number of mold types in each instance was determined based on the number of machines: $t_0 = m + 1$, $t_1 = pn_1$, and $t_2 = pn_2$, where pn_1 and pn_2 are the first and second prime numbers greater than $m + 1$, respectively. For each combination of a machine, jobs, and molds, five instances were generated.

The new instances were grouped into two sets, and Tables 1 and 2 present their characteristics. The number of machines (m), jobs (n), and molds (l); the number of combinations of the previous parameters; the number of instances selected in each combination; and the number of instances proposed are listed in the tables.

The set of smaller instances, RCPMS-I, comprises 90 instances. The instances with 8 jobs have 2 machines, instances with 15 jobs have 2 or 3 machines, and instances with 25 jobs have between 2 and 4 machines. The set of the largest problems, RCPMS-II, comprises 180 instances. The instances with 50 jobs have between 3 and 5 machines, instances with 100 jobs have 4 and 7 machines, and instances with 200 jobs have between 6 and 10 machines.

5.2. Lower bounds

The RCPMS has two trivial lower bounds. First, the value generated by the general model of parallel machines scheduling, which disregards setup times and resource constraints (Pinedo, 2008). Second, the *mold dominance*, i.e., the largest sum value of the processing times of all jobs requiring a specific mold, which may be equal to or greater than the sum of the processing times of all other jobs.

However, these lower bounds are not expected to be tight enough to support an accurate analysis of the new set of instances. Thus, in order to provide optimal solutions and tighter lower bounds, mathematical models from the literature were also considered. Nonetheless, given the limitations of the mathematical models listed in Section 3, we have selected three recent models designed for related problems and used them in addition to the general model of Pinedo (2008) to produce different levels of relaxation to the original RCPMS.

Beezão et al. (2017) presented two models for job scheduling on identical parallel machines with setup times, a relaxed version of the RCPMS which disregards resource constraints. Among the models presented, the model named M_1 showed the best results and was therefore selected. Addressing unrelated parallel machine scheduling with resource constraints and setup times, Bitar et al. (2021) presented a mixed-integer programming (MIP) model to minimize the weighted sum of completion times using time-indexed formulations. To be applied to the RCPMS, the model was adapted to consider identical machines and the objective function was switched to minimization of

Molds	Jobs			
	6	1	0	2
1	0	1	0	0
2	1	0	1	1
3	0	0	0	0

△=14

(a) Mold matrix of the critical machine.

Molds	Jobs			
	6	1	0	2
1	0	1	0	0
2	1	0	1	1
3	0	0	0	0

△=14

(b) Random row selection.

Molds	Jobs			
	1	6	0	2
1	1	0	0	0
2	0	1	1	1
3	0	0	0	0

△=12

(c) First move attempt.

Molds	Jobs			
	1	0	2	6
1	1	0	0	0
2	0	1	1	1
3	0	0	0	0

△=15

(d) Second move attempt.

Fig. 6. 1-block grouping local search.

Table 1
Characteristics of the RCPMS-I set.

m	n	l	Combinations	No. of instances per combination	No. of proposed instances
{2}	{8}	$\{t_0, t_1, t_2\}$	3	5	15
{2, 3}	{15}	$\{t_0, t_1, t_2\}$	6	5	30
{2, 3, 4}	{25}	$\{t_0, t_1, t_2\}$	3	5	45
Total					90

Table 2
Characteristics of the RCPMS-II set.

m	n	l	Combinations	No. of instances per combination	No. of proposed instances
{3, 4, 5}	{50}	$\{t_0, t_1, t_2\}$	9	5	45
{4, 5, 6, 7}	{100}	$\{t_0, t_1, t_2\}$	12	5	60
{6, 7, 8, 9, 10}	{200}	$\{t_0, t_1, t_2\}$	15	5	75
Total					180

the makespan. In a similar context, Ozer and Sarac (2019) presented a MIP to minimize the total weighted completion time for job scheduling on parallel machines with resource constraints, machine eligibility and setup times. This model does not use time-indexed formulations. In order to use this model for the RCPMS, the objective function has changed to minimization of the makespan and machine eligibility constraints have been removed.

All models were implemented using Python 3.7.4, solved using the Gurobi solver 9.1.2 and run with a time limit of 10800 s. The reference values considered in Section 5.6 were defined as solution value or best lower bound value generated by any of the models for each instance. Individual results of each model are available in the Supplementary Material.

The trivial lower bounds were enough to provide optimal results for all instances of the benchmark provided by Chung et al. (2019). For the RCPMS I set, the model of Bitar et al. (2021) reported optimal solutions for 20 instances and the model of Ozer and Sarac (2019) reported optimal solutions for 50 instances, including all solutions found by the first model. The optimal results represent 55.56% of the total for this set, including all instances with 8 and 15 jobs. For the remainder of this set, we considered the highest lower bound among those reported by the three models. Regarding the RCPMS II set, the model of Bitar et al. (2021) proved to be inefficient, being unable to solve even the linear relaxation of the problems. The model of Ozer and Sarac (2019) was able to provide incumbent solutions for instances with 50 and 100 jobs, which can be used as upper bounds on the solution values. Therefore, the lower bounds for the second set of instances were provided by the model of Beezão et al. (2017).

5.3. Upper bounds

As mentioned in Section 3, the best-known results for the only benchmark available in the literature were obtained by combining two list processing heuristics proposed by Chung et al. (2019). Therefore, these heuristics were implemented to provide upper bounds on the solution values of the new benchmark proposed in our study. The original published results of Chung et al. were used as a guideline to validate the implementations, which, following the original description, were implemented using the C++ language.

In order to provide more competitive upper bound values, we additionally present an implementation of the General variable neighborhood search (GVNS) metaheuristic. Introduced by Mladenović and

Hansen (Mladenović and Hansen, 1997), the VNS is a metaheuristic that, from a given initial solution, explores the solution space through systematic exchanges of neighborhood structures within the local search algorithm. The GVNS is an extended version of VNS that uses a VND as a local search.

The steps of our GVNS implementation are described in Algorithm 3. Given an initial solution, s as the input, the main loop (lines 1 to 5) coordinates the application of the metaheuristic until the stopping criterion is met. In the shaking step (line 2), a percentage α of jobs are re-scheduled randomly, generating a neighboring solution s'. In the intensification step, the VND component is applied to s' (line 3). Whenever a neighboring solution with the best solution value is found, the current solution, s, is updated (lines 4 and 5). When the stopping criterion is met, the current solution is returned (line 6). The VND implementation is the same used by the BRKGA, described in Section 4.2. The GVNS coding and validation were performed using the same standards used in the BRKGA development. The value of α and the stopping criterion were defined in the parameter tuning reported in Section 5.4.

Algorithm 3: General variable neighborhood search.

```
   input : solution s
1  while stop_criterion is not satisfied do
2  |  s' ← Shake(s, α);
3  |  s' ← VND(s');
4  |  if C_max(s') < C_max(s) then
5  |  |  s ← s';
6  return s;
```

In addition to the solutions generated by the heuristics and the metaheuristic, we also considered the non-optimal incumbent solutions generated by the models listed in Section 5.2. Thus, the lower bound values considered in Section 5.6 were defined as the best solution value generated by any of these methods for each instance. Individual solution values of each method are available in the Supplementary Material.

For the RCPMS I set, seven instances had the best upper bound provided by incumbent solutions of the model of Ozer and Sarac (2019). For the remaining instances of this set, the best upper was reported by the GVNS metaheuristic. Concerning the RCPMS II set, except for one instance, all upper bounds were achieved by the GVNS metaheuristic. The exception, is an upper bound provided by the model of Ozer and Sarac (2019). Although they are widely used in the related works, the list processing heuristics did not perform well compared to the other methods, as they present an average solution value distance of 50.09% when compared to the GVNS results.

5.4. Parameter tuning

The parameters of the VNS and the hybrid BRKGA were tuned using the offline *irace* method of automatic configuration for optimization algorithms (López-Ibáñez et al., 2016). Given a set of test instances and a set of possible values for each parameter, the irace method determines an appropriate combination of values for them. The set of instances used in this experiment comprises 10% of the new instances, which were randomly selected.

To define the α parameter of the VNS metaheuristic, the irace considered the values {1%, 3%, 5%, 10%, 15%, 20%} and selected 1%. Additionally, considering a trade-off between solution quality and processing time, we defined the stopping criterion as 10000 iterations.

Table 3 presents the options considered by the irace method while defining the parameters of BRKGA. Each option was defined considering the trade-off between solution quality and processing time. The values that were selected for the parameters are highlighted in bold.

To make the best use of the available computational architecture, the decoding and local search step was parallelized. The maximum number of cores used by the hybrid BRKGA was limited to four to avoid discrepancies in the analysis and future comparisons. Following the same rationale, the local search scheme and stopping criteria were defined. To avoid high computing time and premature convergence to non-global local optima, a probability of 5% was defined for the application of local search for each individual. Additionally different strategies for the application of local search to the elite and non-elite sets were adopted. VND was used for the elite group, and local search was performed once per generation for the non-elite individuals. The maximum number of generations was set to 1000, and a trigger was added to interrupt the hybrid BRKGA after 300 generations without improvement in the solution value.

Table 3
Parameter values considered. Selected values in bold.

Parameter	Options
Population size	{2×n, 5×n, **10×n**}
Elite population percentage	{5%, 10%, 15%, 20%, **25%**, 30%}
Mutant population percentage	{5%, **10%**, 15%, 20%, 25%, 30%}
Bias towards elite parents	{60%, 65%, 70%, 75%, 80%, 85%, 90%, **95%**}

5.5. Preliminary experiment

A preliminary experiment was conducted in order to analyze the individual BRKGA components against the hybrid version. This experiment considered the hybrid BRKGA, the original BRKGA without hybridization and the VND alone with random initial solutions. Owing to the randomness present in the methods, we considered ten independent runs and only the best solutions achieved by each method were considered in the analyses.

Considering the hybrid BRKGA as the reference and the RCPMS I set, the original BRKGA reported an average percentage distance of 5.01%, ranging between 0.00% and 28.70%. The VND reported an average percentage distance of 26.04%, ranging between 0.00% and 131.62%. For the RCPMS II set, the BRKGA reported an average percentage distance of 67.94%, ranging between 0.68% and 159.85%. The VND reported an average percentage distance of 126.28%, ranging between 5.87% and 244.54%. Both original BRKGA and VND alone presented solution values worse than the upper bound values, hence, these methods are not considered in the next experiment. The detailed results of this experiment are provided in the Supplementary Material.

5.6. Hybrid BRKGA evaluation

In this experiment, the results generated by the hybrid BRKGA are analyzed considering the benchmarks from the literature and the new benchmark, using as reference the best lower and upper bounds generated by the methods listed in Sections 5.2 and 5.3 .

Regarding the available instances in the literature, the hybrid BRKGA reported optimal results for all instances. Considering the set of small problems, the hybrid BRKGA reported solution values different from the trivial lower bound values given by the solution of the model of Pinedo (2008) only for 10 instances. Nonetheless, the solution to these instances matches a second trivial lower bound, the mold dominance. Considering the set of large problems, the hybrid BRKGA did not report a solution value equal to the first lower bound only for three instances. Again, the solution values for these instances match the second trivial lower bound. Considering all available instances and 10 independent runs, the average running time of the hybrid BRKGA was 19.86 s, demonstrating the ease of the method in the approach of such instances. The average standard deviation was only 0.10, demonstrating the consistency of the hybrid BRKGA in generating solutions with low variation in independent runs.

Table 4
Results for the RCPMS-I set of instances.

m	n	l	Hybrid BRKGA						*Bounds*	
			S^*	S	σ	T	$gap_{LB}(\%)$	$gap_{UB}(\%)$	LB	UB
2	8	3	161.80	161.80	0.00	0.10	0.00	0.00	161.80	161.80
2	8	5	265.00	265.00	0.00	0.11	0.00	0.00	265.00	265.00
2	8	7	287.00	287.00	0.00	0.11	0.00	0.00	287.00	287.00
2	15	3	276.80	278.56	1.80	0.32	0.00	0.00	276.80	276.80
2	15	5	359.80	359.80	0.00	0.29	0.00	0.00	359.80	359.80
2	15	7	330.60	330.96	0.32	0.34	0.00	0.00	330.60	330.60
2	25	3	391.20	391.38	0.29	0.67	0.00	−0.05	391.20	391.40
2	25	5	444.00	444.00	0.00	0.64	0.00	0.00	444.00	444.00
2	25	7	532.60	538.50	6.58	0.75	0.00	0.00	532.60	532.60
3	15	4	201.40	203.96	1.92	0.17	0.23	0.00	201.00	201.40
3	15	5	217.40	219.00	0.92	0.33	0.17	−0.25	217.00	218.00
3	15	7	250.00	251.08	0.38	0.33	0.00	0.00	250.00	250.00
3	25	4	335.20	336.56	4.30	0.04	2.45	0.00	327.60	335.20
3	25	5	333.20	333.92	1.20	0.18	2.88	0.00	323.80	333.20
3	25	7	330.00	335.00	6.18	0.88	9.76	−0.11	301.80	330.40
4	25	5	259.80	265.12	14.09	0.30	10.71	−0.07	235.60	260.00
4	25	7	242.40	253.64	17.75	0.70	16.13	0.00	209.00	242.40
4	25	11	286.00	293.20	7.92	1.05	23.22	−0.57	236.40	287.60

m: number of machines. n: number of jobs. l: number of molds. S^*: best solution values. S: average solution values. σ: standard deviation. T: average running times in seconds. gap: percentage distances between solution values. LB: lower bound. UB: upper bound.

5.6.1. New benchmark

As mentioned in Section 5.1, the set of instances available in the literature proved to be nonchallenging. Therefore, two new sets of instances were presented to assess the quality of the hybrid BRKGA. None of the instances contain dominating molds or can have its jobs easily grouped by molds. There are no previous results for the new benchmark; therefore, the lower and upper bounds on the solution value were used as reference values.

Table 4 presents the average results for the RCPMS-I set, wherein the instances are grouped by the number of machines (m), jobs (n), and molds (l), each group containing five instances. Considering ten independent BRKGA runs for each instance, the table presents the best solution values, (S^*), average solution values (S), standard deviation (σ), and average running times (T) in seconds. Furthermore, the percentage distances (gap) are given between the best hybrid BRKGA solution value and the lower bound on the solution value ($gap_{LB}(\%)$), calculated as $gap_{LB} = \frac{S^* - LB}{LB} \times 100$, and between the best hybrid BRKGA solution and the upper bound ($gap_{UB}(\%)$), calculated as $gap_{UB} = \frac{S^* - UB}{UB} \times 100$. Additionally, the average values of the lower (LB) and upper bounds (UB) are presented.

Regarding the lower bound, the hybrid BRKGA reported an average gap of 3.64%, with a maximum individual gap of 54.92%. The largest differences were observed among the solutions with the largest dimensions. Concerning the instances with proven optimal solution values, the hybrid BRKGA reached 90% of such values. Regarding the upper bound, the hybrid BRKGA reported an average gap of -0.06%, fluctuating between individual gaps of -2.49% and 0.00%. These results demonstrate that, although constituted of small instances, this set is not trivial, working as an initial challenge for future methods.

On average, the hybrid BRKGA required 43.41 generations to improve the initial solutions by 23.22%. The solution quality was the stopping criterion for every run, and it dominated the maximum number of generations. Among the local search procedures, job exchange contributes the most to the solution improvement. On average, job exchange, job insertion, job relocation, and 1-block accounted for 34.63%, 29.31%, 18.5%, and 17.66% of the improvements, respectively. The average running time, 0.41 s and the average standard deviation, 3.54 (equivalent to 0.87%), demonstrate the consistency of the method in generating solutions with negligible running times and low variations over independent runs.

Table 5 presents the average results for the RCPMS-II set, following the same standards of the previous table. The hybrid BRKGA reported an average gap of 31.84% from the lower bound, with a maximum individual gap of 141.86% and a minimum of 0.00%. The largest average gap values are presented for the subgroup: $m = 10$, $n = 200$, $l = 13$, with an average of 99.81%, and the smallest for the subgroup $m = 3$, $n = 50$, $l = 5$ with average gap of 1.62%. It is important to recall that all lower bound values for this set are given by the solution of the model proposed by Beezão et al. (2017), which disregards resource constraints. Regarding the upper bound, the hybrid BRKGA reported an average gap of -11.41%, alternating between individual gaps of 0.27% and -30.00%. The subgroup $m = 8$, $n = 200$, $l = 9$ shows the largest difference with an average gap of -22.76%, and the smallest difference was reported for the subgroups $m = 3$, $n = 50$, $l = 4$ and $m = 3$, $n = 50$, $l = 7$, both with average gap of 0.00%.

On average, the convergence of the hybrid BRKGA occurs after 377.83 generations. In 80.50% of the cases, the execution was interrupted earlier owing to non-improving solutions. The initial solution was improved by 100.77% and the average running time was 155.66 s. Among the local search procedures, job exchange again contributed the most to the solution quality. On average, job exchange, job insertion, 1-block, and job relocation accounted for 30.09%, 27.53%, 24.25%, and 18.13% of the improvements, respectively. Here, 1-block showed an increased contribution. The average standard deviation of 56.76 (equivalent to 9.63%) confirms the robustness of the hybrid BRKGA in generating solutions with low variations over independent runs.

6. Conclusion and future work

In this study, we proposed a new approach to the RCPMS, considering molds as resources. This is an $\mathcal{NP}$-hard problem with practical applications in the microelectronic component industry. The new approach involves the implementation of the BRKGA metaheuristic, hybridized with tailored local search procedures. The proposed method was analyzed using only existing benchmark instances available in the literature and the optimal results were reported for all the instances. A brief analysis revealed the triviality of such benchmark; therefore, 2 new sets comprising 270 problems were proposed to accurately measure the quality of the proposed method and enable the continuity of studies. Comprehensive experiments were performed and considered four mathematical models, two list processing heuristics and one metaheuristic to generate lower and upper bounds used as reference values. Optimal results were published for 55.56% of small instances. Compared to the reference values and considering all new instances, the hybrid BRKGA reported an average gap of 22.44% to the best lower bound values and −7.62% to the best upper bound values. The BRGKA being a metaheuristic that is suitable for permutation problems and given the increase of search effectiveness owing to the use of local

Table 5
Results for the RCPMS-II set of instances.

m	n	l	Hybrid BRKGA						*Bounds*	
			S^*	S	σ	T	$gap_{LB}(\%)$	$gap_{UB}(\%)$	LB	UB
3	50	4	536.00	536.10	0.32	0.22	1.82	0.00	526.40	536.00
3	50	5	578.00	584.14	7.06	3.10	1.62	−0.23	569.40	579.40
3	50	7	595.40	601.54	10.98	3.96	7.71	0.00	553.00	595.40
4	50	5	424.80	440.48	21.17	3.04	7.00	−0.97	397.20	429.00
4	50	7	438.40	460.32	18.20	4.31	11.83	−2.04	392.40	447.60
4	50	11	483.00	504.98	17.48	5.43	31.13	−4.00	369.20	503.80
4	100	5	779.20	806.56	28.75	14.25	8.29	−5.74	716.20	827.60
4	100	7	811.20	856.68	35.91	25.07	13.67	−6.53	713.60	867.40
4	100	11	869.60	919.58	38.17	32.41	20.37	−8.92	723.00	954.80
5	50	6	339.40	390.02	41.66	3.09	15.41	−1.14	294.20	343.20
5	50	7	366.60	413.84	42.14	4.44	17.30	−9.48	312.80	405.60
5	50	11	393.00	422.00	26.32	5.94	28.53	−7.16	306.00	423.20
5	100	6	628.20	704.00	57.13	21.28	12.41	−13.26	559.00	724.40
5	100	7	699.00	743.94	42.47	25.91	19.88	−6.38	538.80	746.60
5	100	11	700.60	777.60	56.08	33.67	21.86	−12.22	575.20	798.60
6	100	7	650.40	732.30	58.23	27.05	32.76	−10.16	489.60	723.80
6	100	11	633.60	183.54	40.12	34.27	32.83	−14.72	476.60	742.60
6	100	13	635.60	713.12	50.78	34.66	31.31	−16.52	484.00	761.60
6	200	7	1165.20	1325.16	113.56	248.13	19.11	−20.98	978.00	1475.60
6	200	11	1126.40	1158.02	27.14	374.47	10.51	−10.75	1019.20	1268.20
6	200	13	1217.40	1325.60	86.49	386.67	23.57	−16.17	985.80	1458.20
7	100	8	614.60	718.24	74.92	34.08	46.52	−13.41	420.00	710.40
7	100	11	589.40	685.18	57.73	39.41	41.01	−16.27	418.60	706.00
7	100	13	587.00	658.22	50.93	40.68	39.50	−15.84	420.60	701.80
7	200	8	1116.40	1281.64	93.16	281.56	30.71	−21.63	855.00	1431.80
7	200	11	1061.60	1245.46	123.32	328.60	25.10	−22.34	849.00	1378.60
7	200	13	1161.80	1276.12	78.42	375.53	38.38	−17.24	840.20	1407.40
8	200	9	1152.60	1407.80	147.10	302.54	57.37	−22.76	733.40	1492.20
8	200	11	1138.20	1284.62	86.47	327.79	54.72	−14.18	736.00	1333.20
8	200	13	1055.80	1226.26	105.05	376.92	44.02	−19.05	734.00	1317.60
9	200	10	1282.40	1481.74	129.20	331.87	89.60	−18.68	677.80	1579.20
9	200	11	907.60	947.30	30.35	358.08	35.93	−9.72	664.80	1008.40
9	200	13	1006.40	1099.56	68.63	364.79	53.64	−13.33	657.20	1172.80
10	200	11	1079.00	1174.14	53.46	320.44	79.53	−10.68	600.00	1216.80
10	200	13	1202.00	1348.04	81.11	348.21	99.81	−16.30	601.80	1436.40
10	200	17	826.60	892.58	43.42	373.99	41.40	−11.76	584.20	945.20

m: number of machines. n: number of jobs. l: number of molds. S^*: best solution values. S: average solution values. σ: standard deviation. T: average running times in seconds. gap: percentage distances between solution values. LB: lower bound. UB: upper bound.

search procedures that consider specific characteristics of the problem addressed, we believe that the presented results are a tighter new upper bound, which is sufficient to provide a challenge for future studies. Our future work will include proving optimal results for the new benchmark and addressing different versions of the RCPMS, such as environments with non-identical machines and a variable number of mold copies.

CRediT authorship contribution statement

Leonardo C.R. Soares: Data curation, Formal analysis, Investigation, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Marco A.M. Carvalho:** Conceptualization, Data curation, Investigation, Methodology, Project administration, Supervision, Writing – original draft, Writing – review & editing.

Acknowledgments

This work was supported by the National Counsel of Technological and Scientific Development (Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq), Brazil 408341/2018-1 and Universidade Federal de Ouro Preto, Brazil. The authors would like to convey their appreciation to Chung Tsuiping, who kindly provided the benchmark instances. This article is dedicated to the loving memory of Luzia Gonçalves de Oliveira Mendes.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cor.2021.105637.

References

A. F. Kummer, N., Buriol, L.S., de Araújo, O.C., A biased random key genetic algorithm applied to the VRPTW with skill requirements and synchronization constraints, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, 2020, pp. 717–724.

Agnetis, A., Alfieri, A., Brandimarte, P., Prinsecchi, P., 1997. Joint job/tool scheduling in a flexible manufacturing cell with no on-board tool magazine. Comput. Integr. Manuf. Syst. 10 (1), 61–68.

Allahverdi, A., 2015. The third comprehensive survey on scheduling problems with setup times/costs. European J. Oper. Res. 246 (2), 345–378.

Andrade, C.E., Ahmed, S., Nemhauser, G.L., Shao, Y., 2017. A hybrid primal heuristic for finding feasible solutions to mixed integer programs. European J. Oper. Res. 263 (1), 62–71.

Bean, J.C., 1994. Genetic algorithms and random keys for sequencing and optimization. ORSA J. Comput. 6 (2), 154–160.

Beezão, A.C., Cordeau, J.-F., Laporte, G., Yanasse, H.H., 2017. Scheduling identical parallel machines with tooling constraints. European J. Oper. Res. 257 (3), 834–844.

Bitar, A., Dauzère-Pérès, S., Yugma, C., 2021. Unrelated parallel machine scheduling with new criteria: Complexity and models. Comput. Oper. Res. 132, 105291.

Bitar, A., Dauzère-Pérès, S., Yugma, C., Roussel, R., 2016. A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. J. Sched. 19 (4), 367–376.

Calmels, D., 2018. The job sequencing and tool switching problem: state-of-the-art literature review. Classif. Trends Int. J. Prod. Res. 1–21.

Chen, J.-F., 2005. Unrelated parallel machine scheduling with secondary resource constraints. Int. J. Adv. Manuf. Technol. 26 (3), 285–292.

Chen, J.-F., Wu, T.-H., 2006. Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. Omega 34 (1), 81–89.

Chung, T., Gupta, J.N., Zhao, H., Werner, F., 2019. Minimizing the makespan on two identical parallel machines with mold constraints. Comput. Oper. Res. 105, 141–155.

Crama, Y., Kolen, A.W.J., Oerlemans, A.G., Spieksma, F.C.R., 1994. Minimizing the number of tool switches on a flexible machine. Int. J. Flexible Manuf. Syst. 6 (1), 33–54.

Gao, L., Wang, C., Wang, D., Yin, Z., Wang, S., 1998. A production scheduling system for parallel machines in an electrical appliance plant. Comput. Ind. Eng. 35 (1–2), 105–108.

Garey, M.R., Graham, R.L., 1975. Bounds for multiprocessor scheduling with resource constraints. SIAM J. Comput. 4 (2), 187–200.

Gonçalves, J.F., Resende, M.G., Toso, R.F., 2012. Biased and Unbiased Random-Key Genetic Algorithms: An Experimental Analysis. AT & T Labs Research, Florham Park.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Annals of Discrete Mathematics. Vol. 5. Elsevier, pp. 287–326.

Guangdong, H., Ping, L., Qun, W., 2007. A hybrid metaheuristic ACO-GA with an application in sports competition scheduling. In: Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing. Vol. 3. SNPD 2007, IEEE, pp. 611–616.

Hong, T.-P., Jou, S.-S., Sun, P.-C., 2009. Finding the nearly optimal makespan on identical machines with mold constraints based on genetic algorithms. In: WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering. No. 8. World Scientific and Engineering Academy and Society, pp. 504–508.

Hong, T.-P., Sun, P.-C., Li, S.-D., A heuristic algorithm for the scheduling problem of parallel machines with mold constraints, in: The 7th WSEAS International Conference on Applied Computer & Applied Computational Science, 2008, pp. 642–651.

Karray, A., Benrejeb, M., Borne, P., 2011. New parallel genetic algorithms for the single-machine scheduling problems in agro-food industry. In: 2011 International Conference on Communications, Computing and Control Applications. CCCA, IEEE, pp. 1–7.

Keung, K., Ip, W., Lee, T., 2001. The solution of a multi-objective tool selection model using the GA approach. Int. J. Adv. Manuf. Technol. 18 (11), 771–777.

Lee, Y.H., Pinedo, M., 1997. Scheduling jobs on parallel machines with sequence-dependent setup times. European J. Oper. Res. 100 (3), 464–474.

Lee, Y.-C., Qi, Q.-F., Hong, T.-P., Chen, C.-H., 2014. A GA-based scheduling algorithm on parallel machines with heterogeneous mounted molds. In: 2014 IEEE International Conference on Granular Computing. GrC, IEEE, pp. 147–150.

Li, D., Wang, J., Qiang, R., Chiong, R., 2020. A hybrid differential evolution algorithm for parallel machine scheduling of lace dyeing considering colour families, sequence-dependent setup and machine eligibility. Int. J. Prod. Res. 1–17.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. Oper. Res. Perspect. 3, 43–58.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Comput. Oper. Res. 24 (11), 1097–1100.

Mokotoff, E., 2004. An exact algorithm for the identical parallel machine scheduling problem. European J. Oper. Res. 152 (3), 758–769.

Ohki, M., Morimoto, A., Miyake, K., 2006. Nurse scheduling by using cooperative GA with efficient mutation and mountain-climbing operators. In: 2006 3rd International IEEE Conference Intelligent Systems. IEEE, pp. 164–169.

Oliveira, B.B., Carravilla, M.A., Oliveira, J.F., Costa, A.M., 2019. A co-evolutionary matheuristic for the car rental capacity-pricing stochastic problem. European J. Oper. Res. 276 (2), 637–655.

OpenMP, 2019. The openmp api specification for parallel programming. https://www.openmp.org/. (Accessed 22 October 2019).

Ozer, E.A., Sarac, T., 2019. Mip models and a matheuristic algorithm for an identical parallel machine scheduling problem under multiple copies of shared resources constraints. Top 27 (1), 94–124.

Paiva, G.S., Carvalho, M.A.M., 2017. Improved heuristic algorithms for the job sequencing and tool switching problem. Comput. Oper. Res. 88, 208–219.

Pinedo, M., 2008. Scheduling: theory, and systems.

Pinheiro, J.C., Arroyo, J.E.C., 2020. Effective ig heuristics for a single-machine scheduling problem with family setups and resource constraints. Ann. Math. Artif. Intell. 88 (1–3), 169–185.

Ramos, A.G., Silva, E., Oliveira, J.F., 2018. A new load balance methodology for container loading problem in road transportation. European J. Oper. Res. 266 (3), 1140–1152.

Reddy, M.S., Ratnam, C., Rajyalakshmi, G., Manupati, V., 2018. An effective hybrid multi objective evolutionary algorithm for solving real time event in flexible job shop scheduling problem. Measurement 114, 78–90.

Soares, L.C.R., Carvalho, M.A.M., 2020. Sequence-dependent setup and machine eligibility, biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints. European J. Oper. Res. 285 (3), 955–964.

Spears, W.M., De Jong, K.D., 1995. On the Virtues of Parameterized Uniform Crossover. Tech. rep., Naval Research Lab, Washington D.C.

Tamaki, H., Hasegawa, Y., Kozasa, J., Araki, M., 1993. Application of search methods to scheduling problem in plastics forming plant: A binary representation approach. In: Proceedings of 32nd IEEE Conference on Decision and Control. IEEE, pp. 3845–3850.

Xianzhang, C., Chengyao, W., 1999. Scheduling on the parallel machines with mold constraint. In: SICE'99. Proceedings of the 38th SICE Annual Conference. International Session Papers (IEEE Cat. No. 99TH8456). IEEE, pp. 1167–1170.

Xiong, H., Fan, H., Li, G., 2013. Genetic algorithm-based hybrid methods for a flexible single-operation serial-batch scheduling problem with mold constraints. Sens. Transducers 155 (8), 232.