Journal Pre-proof

Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints

Leonardo Cabral R. Soares, Marco Antonio M. Carvalho

PII: S0377-2217(20)30199-5
DOI: https://doi.org/10.1016/j.ejor.2020.02.047
Reference: EOR 16367

To appear in: *European Journal of Operational Research*

Received date: 30 June 2019
Accepted date: 27 February 2020

Please cite this article as: Leonardo Cabral R. Soares, Marco Antonio M. Carvalho, Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints, *European Journal of Operational Research* (2020), doi: https://doi.org/10.1016/j.ejor.2020.02.047

Highlights

- We consider the Identical Parallel Machine with Tooling Constraints Problem;
- The single benchmark of instances available were used to evaluate proposed method;
- The generated results are compared to a mathematical model and a metaheuristic;
- Upper bounds were matched (40.76%) or improved (59.10%) on the set of small instances;
- New best solutions were generated for 93.89% of the set of large instances.

Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints

Leonardo Cabral R. Soares[a,*], Marco Antonio M. Carvalho[b]

[a]*Departamento de Computação, Instituto Federal do Sudeste de Minas Gerais, Brazil*
[b]*Departamento de Computação, Universidade Federal de Ouro Preto, Minas Gerais, Brazil*

Abstract

We address the problem of scheduling a set of n jobs on m parallel machines, with the objective of minimizing the makespan in a flexible manufacturing system. In this context, each job takes the same processing time in any machine. However, jobs have different tooling requirements, implying that setup times depend on all jobs previously scheduled on the same machine, owing to tool configurations. In this study, this $\mathcal{NP}$-hard problem is addressed using a parallel biased random-key genetic algorithm hybridized with local search procedures organized using variable neighborhood descent. The proposed genetic algorithm is compared with the state-of-the-art methods considering 2,880 benchmark instances from the literature reddivided into two sets. For the set of small instances, the proposed method is compared with a mathematical model and better or equal results for 99.86% of instances are presented. For the set of large instances, the proposed method is compared to a metaheuristic and new best solutions are presented for 93.89% of the instances. In addition, the proposed method is 96.50% faster than the compared metaheuristic, thus comprehensively outperforming the current state-of-the-art methods.

Keywords: Combinatorial optimization, Flexible manufacturing systems, Metaheuristics, Parallel machines, Scheduling

*Corresponding author
Email addresses: leonardo.soares@ifsudestemg.edu.br (Leonardo Cabral R. Soares), mamc@ufop.edu.br (Marco Antonio M. Carvalho)

1. Introduction

Since the 1980s, manufacturing companies have employed flexible manufacturing systems (FMSs) to handle a wider variety of products and satisfy complex and variable production requirements (Calmels, 2018). A typical FMS comprises a network of flexible machines connected by an automatic material handling system. The flexibility of such systems and machines stems from their capability of performing a large number of different operations such as cutting, drilling, and sanding (Zeballos, 2010). To perform specific operations, different tools must be loaded into the tool magazine of the flexible machine, such as cut blades, drill bits, and sanders. A production line equipped with flexible machines replaces one or more dedicated production lines, enabling industries to have larger product portfolios.

The production scenario addressed in this paper is mainly found in microelectronic and metallurgic companies. In general, each step of product manufacturing is referred to as a *job* to be *processed* by a flexible machine. Each job requires a specific set of tools to be processed. The production is divided into *stages*, at each of which one job is processed. Therefore, the required tools must be loaded into the flexible machine at each production stage. The tool magazine has a limited capacity and is able to store all the necessary tools for each job separately, but it is usually unable to store all the tools required by all jobs simultaneously. Therefore, given a set of different jobs to be processed sequentially, it may be necessary to perform tool switches in the magazine before a different job can be processed.

For tool switches to occur, the machine must be stopped and the tools to be loaded must be moved from storage to the production area. To free space in the magazine, some tools must be unloaded from the flexible machine and sent back to the storage. Then, the new tools can be loaded and the next production stage begins. According to Beezão et al. (2017), the movement of tools between storage and the production area represents between 25% and 30% of the fixed and variable costs in an FMS. Moreover, according to Van Hop &

Nagarur (2004), tool switches consume more time than any other setup, directly affecting the cost of production. As such tool switches are unavoidable in most real FMSs, their minimization becomes critical to increasing productivity.

The production planning process in such an FMS, known as the job sequencing and tool switching problem (SSP), comprises two main combinatorial optimization problems:

1. The sequencing of jobs on each flexible machine.
2. The scheduling of the loading and unloading of tools at each production stage.

The first problem is $\mathcal{NP}$-hard (Crama et al., 1994), meaning that no deterministic polynomial algorithm is known for its solution. Given a fixed sequence of jobs, the optimal scheduling of tool switches can be determined in deterministic polynomial time using the *keep tool needed soonest* (KTNS) policy, introduced by Tang & Denardo (1988).

In this study, we address the SSP in an industrial environment consisting of multiple identical parallel machines, a variation named the identical parallel machines problem with tooling constraints (IPMTC), first addressed by Stecke (1983). The objective is to minimize the makespan, i.e., the completion time of the last processed job. As main contributions, this paper first presents the development of a biased random-key genetic algorithm (BRKGA) hybridized with local search procedures organized using variable neighborhood descent (VND) to solve the IPMTC, which outperforms existing algorithms. Secondly, we publish the results for single benchmark instances available in the literature, which will contribute consistently to the future of the study of the problem.

The remainder of this paper is organized as follows. The IPMTC is formally described in Section 2. Section 3 reviews existing literature on the IPMTC. Next, Section 4 presents the details of the BRKGA metaheuristic, and computational results are reported in Section 5. Finally, we provide conclusions and future research directions in Section 6.

2. Problem description

Consider an FMS containing a set of identical parallel machines $M = \{1, ..., m\}$, with each machine containing a magazine with the capacity for C simultaneous tools; a set of jobs $J = \{1, ..., n\}$, each with a processing time indicated by p_j $(j \in J)$; a set of tools $T = \{1, ..., l\}$; and a subset of tools T_j required to process a job j $(j \in J)$, where $|T_j| \leq C$, with a constant time $\bar{p}$ for any tool switch. The IPMTC consists of determining the assignment of jobs to machines and their sequencing on each machine, such that the makespan Δ is minimized.

From a scheduling perspective, jobs have no release or due time, are not preemptive, and there are no precendence constraints. From a tooling perspective, the formal definition of the IPMTC introduced by Fathi & Barnette (2002) also considers that (i) each machine has a complete set of tools, (ii) a tool can be loaded in any position available in the magazine, and (iii) the setup time required for the initial configuration of the machine is not considered.

According to the classification of Graham et al. (1979), the IPMTC can be stated as $Pm|s_{jk}|C_{max}$, where Pm, s_{jk}, and C_{max} represent the identical parallel machines, sequence-dependent setup times, and makespan, respectively. In contrast to the traditional sequence-dependent setup times, s_{jk} for the IPMTC represents a setup time dependent on the entire job sequencing on a machine, rather than only the jobs j and k. Because the magazine may have a greater capacity than the number of tools required by a single job, tools can be loaded in advance to avoid unnecessary future switches. For example, a tool not required by the job currently being processed may be kept in the magazine if required by a subsequent job. Therefore, all jobs are considered in the sequence-dependent machine setup time.

A solution to an IPTMC instance is given by the sequencing of jobs for each machine. The tool switches on a machine $i \in M$ can be represented by a binary matrix $R^i = \{r^i_{tj}\}$, with $t \in \{1, \ldots, l\}$ and $j \in \{1, \ldots, n\}$. In R^i, the rows represent the available tools and the columns represent the jobs assigned to the machine i in order of processing. An entry $r^i_{tj} = 1$ indicates that the tool t is

loaded on the machine i during the processing of job j, with $r^i_{tj} = 0$ otherwise.

The number of tool switches on a single machine is calculated by Equation (1), as proposed by Crama et al. (1994).

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

The completion time of a job is given by the sum of the job processing times and the time spent on tool switches. Let $x_{ij} = 1$ denote that job j has been assigned to machine i, and $x_{ij} = 0$ otherwise. The IPMTC objective function minimizes the makespan, as shown in Equation (2). The most time-consuming machine is referred to as the *critical machine*.

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

3. Literature review

The IPMTC has been studied since the early 1980s, although work related to scheduling jobs on parallel machines dates back to the late 1950s. Since then, several authors have addressed the IPMTC and several related themes (Allahverdi et al., 1999, 2008; Allahverdi, 2015; Calmels, 2018).

For production management in an FMS, Stecke (1983) listed five production planning problems that would maximize production and reduce costs if solved. The focus was predominantly on the machine *grouping* and *loading* problems. The first problem refers to the partition of a set of non-identical machines into groups of identical machines, while the second refers to the scheduling of jobs and tool switches. The combination of these problems offers a very close definition to that of the IPMTC.

Subsequently, Berrada & Stecke (1986) proposed a branch-and-bound method for the loading problem. The computational experiments considered two sets of 55 instances containing between 4 and 15 identical machines and between 4 and 48 jobs. The proposed method achieved superior results, specifically when the processing times of jobs are heterogeneous.

According to Pinedo (2018), a *flexible flow shop* is a generalization of the Pm environment. Considering this environment, Koulamas (1991) presented the *reducing tool requirements* algorithm and the *efficient search technique* to minimize the number of tool replacements at each production level owing to breaks without increasing the makespan. To increase tool life, the processing of jobs not assigned to critical machines is slowed down, reducing tool wear. The computational experiment employed ten instances with 4, 8, and 15 levels of production and 50 jobs. The experiment produced optimal solutions within a short processing time.

Another machine environment considered to be a generalization of Pm is the *job shop* (Pinedo, 2018). Hertz & Widmer (1996) considered a job shop with tooling constraints, and presented a tabu search method with the goal of minimizing the makespan. The computational experiments considered eight problems, with the numbers of jobs and machines ranging from 1 to 10. The results were compared with an earlier version of the tabu search presented in Widmer (1991), and the later version obtained better results for most instances.

The study in Agnetis et al. (1997) considered a restricted machine environment for the IPMTC, where $m = 2$, $C = 1$, and tools are shared between machines. This tool sharing implies a tool availability constraint, i.e., a specific tool cannot be loaded in both machines at the same time. A tabu search was employed with the aim of minimizing the makespan.

The IPMTC has predominantly been considered in its *offline* version, where all jobs are known a priori and have release times of zero. In contrast, Kurz & Askin (2001) also addressed the *online* version, where jobs are not known in advance and are released at different points in time. Both versions were modeled as traveling salesman problems and solved using three different heuristics and a genetic algorithm. However, the obtained results were only compared to each other, precluding a substantial analysis.

Fathi & Barnette (2002) formalized the current definition of the IPMTC, and addressed the problem using list-processing heuristics and local search procedures based on the moves of insertions and exchanges of jobs between different

machines. The computational experiments considered three groups of randomly generated instances, and compared the results of the different methods to two lower bounds on the makespan value. Two descent search methods based on the local search procedures, named CLIP_1 and CLIP_2, generated the best results.

The work of Fathi & Barnette (2002) remained the state-of-the-art for more than a decade and a half. During this period, to the best of our knowledge, no work addressing the standard IPMTC was published. Notwithstanding, Beezão et al. (2017) recently studied the IPMTC and proposed two integer linear programming models and two versions of the adaptive large neighborhood search (ALNS) metaheuristic.

The first and second models, named M_1 and M_2, were inspired by the models of Tang & Denardo (1988) and Laporte et al. (2004) for the SSP, respectively. The ALNS principle involves combining a set of insertion and removal heuristics to generate numerous neighborhoods of a current solution. Nine removal operators and five insertion operators were developed, some based on the traditional sequence-dependent setup time and others based on the IPMTC characteristics. The ALNS versions differ in how the initial solutions are generated: the ALNS-R version generates random initial solutions, while ALNS-B employs a heuristic designed for the SSP and divides the resulting single schedule of jobs into m schedules using an integer linear programming model.

The computational experiments considered two sets of 1440 instances each, named IPMTC-I and IPMTC-II. These instances are described in Section 5. The formulations M_1 and M_2 were evaluated considering a small group of instances from the IPMTC-I set. Both formulations solved all instances with eight jobs, and could not solve most instances with 15 or 25 jobs. According to the authors, the M_1 model obtained better overall results. The ALNS versions were compared with each other as well as the CLIP_1 and CLIP_2 methods (Fathi & Barnette, 2002), in a new implementation performed by the authors. As in the previous experiment, the solution values of the compared methods were not reported. Instead, the percentage distances from the best solution found in the experiments and the processing times were reported in the pa-

per. According to the presented data, both versions of the ALNS outperformed CLIP_1 and CLIP_2 in all instances of the IPMTC-I set. For the instances of the IPMTC-II set, ALNS-B faced difficulties in all instances, and reached the time limit of 3600 seconds before even determining the initial solution for most instances. ALNS-R once more outperformed CLIP_1 and CLIP_2 in all instances of this set. Therefore, this constitutes the current state-of-the-art for the IPMTC. In the computational experiments reported in Section 5, we consider the same instances and compare our method with ALNS-R.

The study reported in Gökgür et al. (2018) addressed the scheduling of jobs on unrelated parallel machines. A constrained programming formulation was presented to minimize the makespan. However, tool switching times were not considered, and each tool was considered to have a limited number of copies, implying tool sharing and eventual tool unavailability. The proposed formulation performed better than a mixed integer linear programming model and a previous tabu search implementation for large instances. More recently, Chung et al. (2019) considered the process of wafer production in semiconductor plants, which represents a restricted case of the IPMTC where there are two machines and tools are shared. A mathematical formulation and three heuristics were presented to approach the problem, which outperformed previous metaheuristics from the literature.

4. Methods

Our approach to the IPMTC is based on the evolutionary metaheuristic biased random-key genetic algorithm, or BRKGA, of Gonçalves & Resende (2011). This was chosen owing to its suitability for permutation problems, such as the IPMTC, and its recent strong performances on relevant combinatorial problems (Andrade et al., 2017; Ramos et al., 2018; Oliveira et al., 2019). In addition, we believe that the use of tailored components contributes to an improved metaheuristic performance. Considering the BRKGA in particular, we propose a hybridization with variable neighborhood descent, or VND, of Mladenović &

Hansen (1997), thereby assisting the intensification of the search by performing a systemic local search and is thus suitable for combined use. In the following sections, we describe the basics of BRKGA and VND and provide details of our implementations.

4.1. Biased random-key genetic algorithm

A random key genetic algorithm (RKGA) (Bean, 1994) is a variant of the traditional genetic algorithm, particularly suitable for permutation problems such as sequencing jobs. To this end, *individuals* (potential solutions) are encoded as vectors of randomly generated real numbers (or keys) in the continuous interval [0, 1]. The RKGA follows a common framework of the genetic algorithm: a *population* of *pop* randomly generated individuals is evolved over a number of *generations* (iterations). A *fitness* value is computed for each individual, and a small group of pop_e individuals with the best fitness values are considered as the *elite*. The remaining $pop - pop_e$ individuals are considered *non-elite*. The population of the generation $g + 1$ is formed from the population of the generation g through the application of selection, reproduction, and mutation operators.

In particular, the RKGA copies the elite individuals from the generation g to $g + 1$, introduces pop_m new individuals generated randomly as *mutants*, and generates $pop - pop_e - pop_m$ *offspring* individuals by combining two *parent* individuals selected randomly from g using the *parametrized uniform crossover* (Spears & De Jong, 1995), in which the i-th keys of an offspring are copied from one of the parents with equal probability. The only problem-dependent component of the RKGA is the *decoder*, which transforms a vector of random keys into a solution for a particular optimization problem.

Inspired by the RKGA, Gonçalves & Resende (2011) presented the BRKGA. This differs from RKGA in the manner in which parents are selected and how the crossover is implemented. While both parents in the RKGA are chosen randomly from the entire population, in the BRKGA one parent is always chosen randomly from the elite group, while the other is chosen randomly from the non-elite group. Again, the parametrized uniform crossover is employed. However,

in the BRKGA the offspring is more likely to inherit the genes of the parent individual belonging to the elite group.

4.1.1. Encoding and decoding

To represent an IPMTC solution, keys are generated in the interval $[1, m+1) \in \mathbb{R}$. The number of keys in each individual is defined by the number n of jobs. The rationale behind this encoding is that the integer part $i \in M$ of the random key indicates the machine i to which each job will be allocated. The decimal part of the key is useful in sequencing (sorting) jobs in a specific machine i. Figure 1(a) illustrates the encoding of an individual for an IPMTC instance containing two machines and six jobs. An individual is decoded into a valid IPMTC solution by sorting its keys and preserving their indices in non-decreasing order. The individual from Figure 1(a) is sorted in Figure 1(b). Thus, this example corresponds to the assignment of jobs 3 and 0 to machine 1 and jobs 2, 4, 1, and 5 to machine 2, in that order.

a)

Index	0	1	2	3	4	5
Genes	1.3256	2.9863	2.0007	1.1258	2.1456	2.9971

b)

Original index	3	0	2	4	1	5
Genes	1.1258	1.3256	2.0007	2.4156	2.9863	2.9971

Figure 1: Encoding of an individual for instance with two machines and six jobs.

The fitness of an individual is the makespan calculated according to Equation (2) after the decoding step. The sum of the processing times of the jobs is trivial, and is performed in linear time in the number of jobs. The scheduling of the loading and unloading of tools on machine i requires an examination of matrix R^i and can be determined in time $O(nl)$ using the KTNS (Tang & Denardo, 1988) policy. This determines that in the case of a tool switch, tools that will no longer be utilized must be removed from the magazine. If it is necessary to remove a tool that will be used again, then the tools that will take longer to be reused must be removed.

4.1.2. Hybridization

The proposed hybridization is performed in the decoding step, and is applied exclusively to elite individuals. After an actual solution is obtained from an individual, a VND is applied to it as a second component of search intensification. If the resulting solution changes, then it is encoded back as an individual that represents the solution, and a new fitness value is assigned to it. After the individuals have all been processed and re-encoded, the BRKGA execution is resumed. The solution for an IPMTC instance is represented by the individual with the lowest fitness value at the end of the evolutionary process.

Algorithm 1 presents the pseudocode for the hybrid BRKGA. The input consists of the number of jobs (n), number of machines (m), population size (pop), size of the elite group (pop_e), number of mutant individuals (pop_m), and maximum number of generations (max_{gen}). First, the initial population is randomly generated (line 1). Then, the main loop (lines 2 to 21) evolves the population over max_{gen} generations. Each new individual is transformed into an actual solution to the IPMTC (line 3), which is evaluated (line 4), and if it is an elite individual then a VND is performed (line 5). All decoded solutions are encoded back as individuals, their fitness are updated (line 6) to reflect the VND changes, and a new elite group is formed (lines 7 and 8). The population of the next generation is formed so far by the newly identified elite group (line 9) and the mutant individuals (line 10). The inner loop (lines 11 to 20) generates the remaining individuals by randomly selecting two parents, one from the elite group (line 12) and one from the non-elite group (line 13). Then, the offspring individual inherits each one of its genes (lines 14 to 19) either from the elite parent (line 17), with probability ρ (line 15), or from the non-elite parent (line 19). Each newly generated individual is added to the next generation (line 20). After its completion, the next generation replaces the previous one (line 21), and the process is repeated until the stopping criterion is met.

Algorithm 1: Hybrid biased random-key genetic algorithm

input : n, m, pop, pop_e, pop_m, max_{gen}

Generate population Π with pop vectors of n random keys chosen from $[1, m+1)$;
while max_{gen} *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 solutions;
    Encode each solution as individuals and update their fitnesses;
    Sort individuals in non-decreasing fitness order;
    Group the first pop_e individuals in Π_e;
    Initialize the population of the next generation $\Pi^+ \leftarrow \Pi_e$;
    Generate a set of pop_m mutants and add it to Π^+;
    for $i \leftarrow 1$ **to** $pop - pop_e - pop_m$ **do**
        Select a random individual a from the elite group;
        Select a random individual b from the non-elite group;
        for $j \leftarrow 1$ **to** n **do**
            Throw a biased coin with probability ρ of heads;
            if *heads* **then**
                $c[j] \leftarrow a[j]$;
            else
                $c[j] \leftarrow b[j]$;
            $\Pi^+ \leftarrow \Pi^+ \cup \{c\}$;
    $\Pi \leftarrow \Pi^+$;

4.2. Variable neighborhood descent

For a given solution s, a local search procedure defines a *neighborhood* composed of a set of solutions with similar characteristics. This is achieved by applying a *move*, i.e., an operation that modifies one or more elements of s, generating a solution s'. If s is updated at each new best solution found, then

the local search is said to be a *descent* search. If no best solution value is found in a neighborhood, then s represents a *local optimum* in the neighborhood structure. The best solution value considering all possible neighborhood structures is a *global optimum*.

The VND, proposed by Mladenović & Hansen (1997), is a metaheuristic consisting of systematic changes of neighborhood structures during the exploration of the solution space. The rationale is that a local optimum shared by different neighborhoods is more likely to be a global optimum than one associated to a single neighborhood. Given a solution, different neighborhoods are explored sequentially. Whenever a new best solution is found, the method restarts the exploration from the first neighborhood. If a solution is a local optimum and no improvement is possible in a neighborhood, then the method proceeds to the next neighborhood. The method ends when all neighborhoods have been explored and a better solution is not found. In this manner, a local optimum common to all neighborhoods is returned.

In our implementation, we employ three different local search procedures: job insertion, job exchange, and 1-block grouping. The first two are traditional methods in the job scheduling literature, while the third is a recent method tailored to the structure of the SSP and IPMTC. Algorithm 2 presents the pseudocode for the VND. The input consists of a feasible solution s. The loop (lines 2 to 13) explores the neighborhoods sequentially: job insertion (lines 3 and 4), job exchange (lines 5 and 6), and 1-block grouping (lines 7 and 8). Each exploration results in a local optimum s', whose makespan is evaluated and compared to the previous one (line 9). If it is smaller, then s is updated and the first neighborhood is explored again (lines 10 and 11). Otherwise, the VND moves to the next neighborhood (line 13). In the end, a local optimum s that is common to all neighborhoods is returned.

Algorithm 2: Variable neighborhood descent

```
input : solution s
k = 1;
while k ≠ 4 do
    if k = 1 then
        s' ← JobInsertion(s);
    if k = 2 then
        s' ← JobExchange(s);
    if k = 3 then
        s' ← OneBlocksGrouping(s);
    if Δ (s') < Δ (s) then
        s ← s';
        k ← 1;
    else
        k ← k + 1;
return s;
```

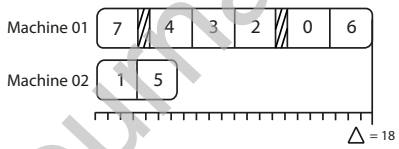
The order of application of the local search procedures was defined with the aim of most effectively exploiting the intrinsic characteristics of each. Job insertion is performed first, which has a tendency to reduce the makespan with a greater emphasis, resulting in balanced job assignments. Next, job exchange is applied, which results in tighter overall job assignments. Finally, we focus on tool switches, which are sometimes neglected. Specifically, it is sought to minimize tool switches on each isolated machine, resulting in an even tighter schedule. This order also benefits the BRKGA running time, as the more complex moves are performed fewer times.

4.2.1. Job insertion

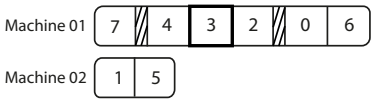
The local search attempts to reduce the makespan of a solution by moving jobs from the critical (i.e., most time-consuming) machines to the least time-consuming machines. To this end, the m machines are sorted in non-decreasing

processing time order, such that machine 1 is the least time-consuming and m is the critical one. Jobs are then randomly selected one at a time from machine m. If the difference between the processing times of these machines is less than the processing time of the selected job, then the move is discarded. Otherwise, the selected job is removed from machine m and assigned to machine 1, in a position that results in the least number of additional tool switches. If the resulting processing time of machine 1 is smaller than the makespan, then the machines are sorted again, the makespan is updated, and the process restarts. After all jobs from the critical machines have been considered and no improvement in the solution value is possible, the solution is a local optimum and the local search procedure finishes.

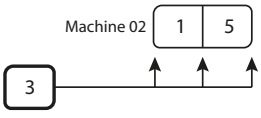
Figure 2 illustrates a hypothetical job insertion. In (a), we have a feasible solution composed of two machines with $\Delta = 18$, where the hatched area indicates the setup time. A random job is selected from the critical machine (b). Then, all possible positions for insertion in machine 02 are evaluated (c), and the one adding the least number of tool switches is selected (d), generating a neighbor solution with $\Delta = 15$.



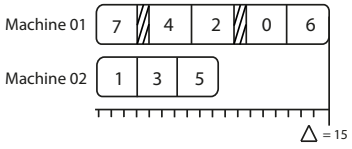
(a) Initial solution.



(b) Random job of the critical machine.



(c) Insertion in the best position.



(d) Neighbor solution.

Figure 2: Job insertion local search.

4.2.2. Job exchange

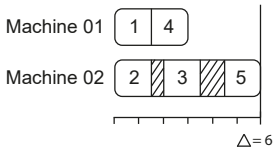
Exchanging two jobs between critical and non-critical machines may reduce the makespan if either these jobs have different processing times or the resulting number of tool switches is smaller. To this end, the m machines are sorted in non-decreasing processing time order, such that machine 1 is the least time-consuming machine and machine m is the critical one. Let J_1 and J_m represent the jobs assigned to machines 1 and m, respectively. For each job $j \in J_m$, each job $k \in J_1$ is randomly selected as a candidate for exchange. This neighborhood size is reduced by applying the minimum compatibility policy proposed by Fathi & Barnette (2002): both machines involved in the exchange must have at least 50% of tools in common considering their assigned jobs, except for the two considered in the exchange.

If the machines are compatible, then jobs j and k are exchanged. If the resulting processing times of machines 1 and m are smaller than the makespan, then the machines are sorted again, the makespan is updated, and the process restarts. If the machines are not compatible, then the move is discarded and a different job $k \in J_1$ is considered. If no improvement is possible involving job j, a different job in J_m is selected and the procedure is repeated. After all pairings of jobs from J_1 and J_m have been considered and no improvement in the solution value is possible, the solution is a local optimum and the local search procedure finishes.

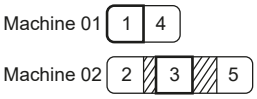
Figure 3 illustrates a hypothetical exchange of jobs. In (a), we have a tool matrix used to calculate the minimum compatibility policy. In (b), a feasible solution is presented, which consists of two machines with $\Delta = 6$. The hatched area indicates the setup time. Two jobs are randomly selected (c) and evaluated concerning their tool compatibility with the other machines. As $T_3 = \{2, 3\}$ and $J_{01} = \{1\}$, the minimum compatibility is not satisfied, and the move is discarded. Next, a new job on machine 01 is selected (d), which has the required tool compatibility. Then, the exchange is performed, generating a neighbor solution with $\Delta = 5$.

Tools	Jobs				
	1	2	3	4	5
1	1	0	0	1	1
2	1	0	1	0	0
3	0	1	1	0	0
4	0	1	0	0	1

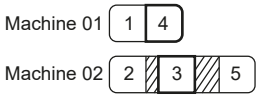
(a) Tool matrix.



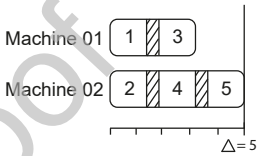
(b) Initial solution.



(c) First selection of jobs.



(d) Second selection of jobs.



(e) Neighbor solution.

Figure 3: Jobs exchange local search.

4.2.3. Grouping of 1-blocks

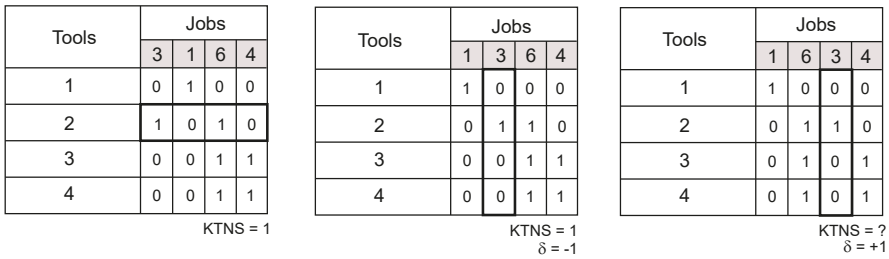
As mentioned in Section 2, the tool switches on a machine $i \in M$ can be represented by a binary matrix R^i, where an entry $r^i_{tj} = 1$ indicates that a tool t is loaded on the machine i during the processing of job j, with $r^i_{tj} = 0$ otherwise. Therefore, an inversion from 0 to 1 in any row of R^i indicates a tool switch. A *1-block* (Crama et al., 1994) is a contiguous sequence of nonzero elements in a row of a binary matrix, which may be enclosed by zero elements. Considering the analogy between the number of 1-blocks on the same row of a matrix and the number of switches of a particular tool in R^i, Paiva & Carvalho (2017) proposed the *1-block grouping* local search for the SSP. This local search attempts to reduce the number of 1-blocks in each row of R^i by grouping its nonzero elements. Thus, it may reduce the number of tool switches on a machine.

To this end, the m machines are sorted in non-decreasing processing time order, such that machine 1 is the least time-consuming machine and machine m is the critical one. The matrix R^m is analyzed row by row in random order, searching for two or more 1-blocks. For each pair of such blocks, the algorithm

moves the columns of the first block one-by-one to the left or right of the second block, to group them together. Each move is analyzed using the δ-evaluation described below. If there is an increase in the overall number of 1-blocks of R^m, then the move is discarded. Otherwise, a second evaluation is performed using the KTNS policy, and all non-degrading moves are accepted. In the case of a tie between solution values generated by the moves to the left and right of the second 1-block, the last evaluated move is kept. If the resulting processing time of machine m is smaller than the makespan, then the machines are sorted again, the makespan is updated, and the process restarts. After all rows from R^m have been considered and no improvement in the solution value is possible, the solution is a local optimum and the local search procedure finishes.

Once a solution evaluated previously by the complete evaluation function has been changed, it may be possible to evaluate the isolated impact of this change and check whether the solution value has improved or degraded. In the context of consecutive block minimization, Haddadi et al. (2015) presented a δ-evaluation that can verify the impact in $O(l)$ time on the number of 1-blocks after a single column is moved from its original position or when two columns exchange positions. Specifically, in the IPMTC context, reducing the number of 1-blocks does not necessarily result in an improved solution. Owing to the use of the KTNS policy, a tool that is not employed on a production stage may be kept in the magazine for future use, and is not necessarily switched. However, increasing the number of blocks makes it impossible to improve the value of the solution.

Figure 4 illustrates a hypothetical grouping of 1-blocks. In (a), we have the critical machine tool matrix, the number of tool switches, and a randomly selected row with two 1-blocks of size one. Moving the first 1-block to the left of the second 1-block (b) decreases the number of 1-blocks, as indicated by the δ-evaluation but does not decrease the number of tool switches, as indicated by the KTNS policy. On the contrary, moving it to the right of the second 1-block (c) increases the number of 1-blocks, and thus the move is discarded and the solution remains unchanged.



(a) Random row selection. (b) Movement to the left. (c) Movement to the right.

Figure 4: 1-block grouping local search.

5. Computational experiments

A set of computational experiments were performed to tune the hybrid BRKGA components and compare it fairly with the state-of-the-art methods. The next sections describe the benchmark instances considered and report the experimental results regarding parameter tuning and comparison with the state-of-the-art methods. Detailed results for each experiment can be found in the supplementary material.

The computational environment adopted for the computational experiments consisted of a computer with an Intel Xeon E5-2660 2.2 GHz processor with 384 GB of RAM under the CentOS 6.8 operating system and an individual thread score of 1.499 (Passmark, 2018). The BRKGA was implemented in C++, and compiled using GCC 4.8.4 and options -O3 and *-march = native*. The OpenMP (2019) shared-memory parallel programming API version 5.0 was used to parallelize the decoding step.

5.1. Instances

The 2880 benchmark instances provided by Beezão et al. (2017) are considered. The instances are separated into two sets, IPMTC-I and IPMTC-II, each containing 1440 instances. Table 1 presents the characteristics of each set. The column headings represent the numbers of instances (#), machines (m), jobs (n), and tools (l).

Table 1: Characteristics of the instances.

set	#	m	n	l
IPMTC-I	240	2	{8, 15, 25}	{15, 20}
	480	3	{15, 25}	{15, 20}
	720	4	{25}	{15, 20}
IPMTC-II	360	{3, 4, 5}	50	{30, 40}
	480	{4, 5, 6, 7}	100	{30, 40}
	600	{6, 7,8, 9, 10}	200	{30, 40}

5.2. Parameter tuning

The parameters of the 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 of the parameters, irace determines an appropriate combination of values for them. In preliminary experiments we considered different values for the parameters, including those recommended by Gonçalves & Resende (2011), and narrowed down the options to the values presented in Table 2, which also presents the values selected by irace.

Table 2: Values considered and selected by to define the parameters.

Parameter	Options	Selected value
Population size (pop)	{2×n, 5×n, 10×n}	5×n
Elite population portion (pop_e)	{10%, 15%, 20%, 25%, 30%}	30%
Mutant population portion (pop_m)	{10%, 15%, 20%, 25%, 30%}	25%
Bias towards elite parents (ρ)	{60%, 65%, 70%, 75%, 80%, 85%, 90%}	85%

Additionally, the VND particulars and BRKGA stopping criterion were defined empirically to achieve a balance between the solution quality and computational effort. To avoid premature convergence and long running times, the VND is only applied to elite individuals, and is not applied to individuals that have not changed since its last application, as these represent local optima. The maximum number of BRKGA generations is set to 100, and the maximum running

time limited to 3600 seconds. Finally, the BRKGA decoding step is particularly suitable for parallel computing (Gonçalves & Resende, 2011). Therefore, the decoding and VND applications were parallelized. To avoid excessive disparity in the comparisons, the maximum number of threads used by BRKGA was limited to four.

5.3. Comparison with the state-of-the-art

The original experiments reported in Beezão et al. (2017) compared ALNS-R, ALNS-B, CLIP_1, and CLIP_2, among other methods, but the solution values were not presented. Instead, the percentage distances from the best solution found in the experiments and processing times were reported. The original ALNS-R results available on the Mendeley Data (Beezão, 2019), covers the IPMTC-II set only, preventing a comparison regarding the IPMTC-I set results.

As mentioned in Section 3, Beezão et al. (2017) also proposed two mathematical formulations, whose results could be used in benchmarking. However, an unspecified subset of instances was considered and optimal solution values were not presented. Therefore, in an attempt to obtain optimal solution values and utilize these as a reference for the BRKGA results on the IPMTC-I set, we implemented M_1, the best performing model according to Beezão et al. (2017). The model was implemented in *Python* version 2.7.12 and solved using *Gurobi* version 8.0.1. The original limit of 4200 s for running time was adopted.

Table 3 presents the results for the IPMTC-I set. Instances are grouped by the numbers of machines (m), jobs (n), and tools (l). 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), all given in seconds. In addition, the percentage distances ($gap(\%)$) are given between the best BRKGA solution value and the solution values generated model M_1 (UB), calculated as $gap = \frac{S^* - UB}{UB} \times 100$. Proven optimal solutions generated by model M_1 are marked with *, while all others are upper bounds on the solution value. Bold values identify the best results for each group of instances.

Table 3: Results for the IPMTC-I set of instances.

m	n	l	BRKGA					M_1
			S^*	S	σ	T	$gap(\%)$	UB
2	8	15	**227.49**	227.50	0.02	0.03	0.00	**227.49***
2	8	20	**250.70**	250.83	0.30	0.02	0.00	**250.70***
2	15	15	**367.22**	368.04	0.95	0.48	-5.82	390.77
2	15	20	**370.08**	370.65	0.63	0.55	-3.64	384.06
2	25	15	**454.10**	454.14	0.06	2.87	-1.37	460.38
2	25	20	**510.24**	510.33	0.10	3.92	-4.42	533.81
3	15	15	**230.53**	231.24	0.82	0.20	-8.33	251.48
3	15	20	**217.92**	218.50	0.55	0.22	-5.45	230.49
3	25	15	**270.00**	271.30	1.48	1.05	-2.57	277.13
3	25	20	**306.48**	307.45	0.88	1.40	-8.05	333.32
4	25	15	**189.99**	190.29	0.30	0.56	-0.63	191.20
4	25	20	**200.83**	204.12	2.59	0.71	-8.05	218.41

* proven optimal solutions

For instances with eight jobs the model M_1 achieved all the optimal results. However, the model has already faced difficulties for larger instances, starting from instances with 15 jobs. For the IPMTC-I set, the model found optimal results for 24.31% of the instances. Only one of these optimal solution values was not matched by the BRKGA. New best upper bounds were found for 851 instances (or 59.10%) of this set. If we consider better or equal results, then the proposed method obtained the best results for 1438 instances (or 99.86%). The largest differences were observed among the solutions for the group $m = 3, n = 15, l = 15$. The mean gap was -3.20%, with a maximum individual gap of 0.59% and minimum of -28.30%. The average standard deviation of 0.72 for the BRKGA demonstrates the consistency of the method in generating solutions with low variations over independent runs.

On average, the VND component improved the quality of solutions of the

IPMTC-I set by 0.43%. Among the local search procedures, job insertion contributes most to the solution quality. On average, job insertion accounts for 60.91% of the improvements, job exchange for 27.19%, and 1-block grouping for 11.90%.

A statistical analysis was performed to compare the two methods. The Shapiro–Wilk normality test (Shapiro & Wilk, 1965) confirmed the null hypothesis that the results compared from BRKGA ($W = 0.88405$, *p-value* = 0.09879) and M_1 ($W = 0.89268$, *p-value* = 0.1276) could be modeled according to a normal distribution, with a confidence interval of 95%. The parametric *Student's t-test* (Student, 1908) was applied to verify whether there is a significant difference between the compared results. The test indicated that there is a significant difference between the results of the methods for the instances of the IPMTC-I set, and that BRKGA has the best mean values ($t = -4.5271, df = 11, p = 0.0004309$) with a significance level of 0.05.

The average running time of BRKGA for all instances of this set was 1.01 s. The maximum running time was 11.46 s, suggesting the efficiency of the method in approaching such instances. For model M_1, the average running time was 3186.03 s. In 1090 instances (75.69% of this set) the time limit of 4200 s was reached.

For the instances of the IPMTC-II set, model M_1 could not even solve the linear relaxation within the time limit. Therefore, we compare the BRKGA results to the original ALNS-R results (Beezão, 2019). The original ALNS-R settings consider a maximum of 25000 iterations, a limit of 14400 s on the running time and the number of removed jobs is 25% of the total. Table 4 presents the average results for 10 independent runs of the ALNS and BRKGA for the IPMTC-II set, following the same standards of the previous table. In order to provide a minimal comparison on running times, the presented BRKGA values were converted using a factor of 1.20, based on the Passmark (2018) benchmark of both computational environments.

The BRKGA generated new best solutions for 1352 instances (93.89%) of this set. However, on two groups of instances the average gap of 0.63% indicates

Table 4: Results for the IPMTC-II set of instances.

m	n	l	BRKGA					ALNS			
			S^*	S	σ	T	$gap(\%)$	S^*	S	σ	T
3	50	30	**1336.73**	1365.00	16.03	87.75	-0.16	1338.85	1364.74	14.01	11237.29
3	50	40	1518.57	1536.11	9.97	118.57	0.84	**1505.92**	1529.15	12.31	13979.03
4	50	30	**996.30**	1019.90	12.79	40.42	-1.24	1008.85	1030.62	11.47	11705.45
4	50	40	1039.13	1057.58	10.56	53.78	0.42	**1034.80**	1059.30	12.29	14311.55
4	100	30	**2179.80**	2208.43	15.99	185.27	-5.97	2318.13	2376.55	33,72	14433.29
4	100	40	**2698.33**	2733.59	19.10	297.34	-7.41	2914.42	2984.09	43.92	14480.79
5	50	30	**809.83**	828.44	9.68	21.83	-0.55	814.32	831.25	9.12	10535.76
5	50	40	**775.18**	791.89	10.18	27.99	-0.02	775.33	792.93	9.88	14044.71
5	100	30	**1743.83**	1766.46	12.56	89.71	-6.99	1874.95	1921.41	25.92	14446.20
5	100	40	**2237.43**	2269.39	16.67	154.31	-8.12	2435.08	2494.69	35.54	14482.06
6	100	30	**1497.82**	1520.69	12.77	49.78	-6.44	1600.85	1643.21	23.63	14431.45
6	100	40	**1930.17**	1958.68	15.87	84.58	-7.91	2095.85	2158.14	35.37	14485.33
6	200	30	**2932.23**	2961.92	17.04	1780.59	-8.74	3212.97	3291.48	48.11	14524.62
6	200	40	**3539.23**	3574.86	19.09	2524.16	-8.25	3857.63	3988.87	77.46	14633.74
7	100	30	**1237.33**	1257.34	10.67	29.67	-5.99	1316.23	1346.65	17.29	14428.72
7	100	40	**1749.20**	1779.50	15.41	52.93	-4.18	1825.48	1869.58	23.43	14412.62
7	200	30	**2410.75**	2438.57	14.50	1078.50	-8.21	2626.37	2709.82	49.69	14541.81
7	200	40	**3178.30**	3217.27	20.06	1711.91	-8.27	3465.02	3586.36	68.74	14622.22
8	200	30	**2245.20**	2273.57	15.61	658.84	-8.28	2448.02	2517.24	43.00	14534.03
8	200	40	**2752.63**	2788.63	17.21	1011.76	-8.95	3023.32	3118.65	59.88	14624.44
9	200	30	**1880.13**	1902.56	12.16	410.69	-7.86	2040.53	2102.57	37.30	14531.10
9	200	40	**2346.20**	2373.00	14.27	678.48	-8.27	2557.75	2647.69	55.07	14637.45
10	200	30	**1827.12**	1849.48	12.22	297.65	-8.45	1995.75	2049.91	33.44	14528.05
10	200	40	**2059.75**	2086.52	13.96	466.93	-8.88	2260.57	2332.38	41.63	14637.09

a slightly inferior performance than the ALNS. The average gap considering all 1440 instances is -4.90%, with an individual maximum gap of 5.43% and a minimum of -14.07% on one of the largest instances. Grouping the instances by the number of jobs, the average gaps for instances with 50, 100, and 200 jobs are -0.12%, -8.84%, and -8.36%, respectively. The average standard deviations of 14.98 (or 0.77%) for the BRKGA and 34.30 (or 1.66%) for the ALNS demonstrate the consistency of the methods in generating solutions with low variations over independent runs.

The VND improved the quality of solutions of this set by 11.05% on average.

Among the employed local search procedures, job insertion (68.93%) again contributes most to the quality of the solution, followed by job exchange (27.61%) and 1-block grouping (3.47%).

A statistical analysis was performed to further compare the methods. The Shapiro–Wilk normality test (Shapiro & Wilk, 1965) confirmed the null hypothesis that the BRKGA ($W = 0.97884$, $p\text{-}value = 0.8736$) and ALNS ($W = 0.97746$, $p\text{-}value = 0.8448$) results could be modeled according to a normal distribution with a confidence interval of 95%. The parametric *Student's t-test* (Student, 1908) was applied to verify whether there is a significant difference between the compared results. The t-test indicated that there is a significant difference, and that BRKGA has the best mean values $t = -6.7602, df = 23, p = 3.391e-07$) with a significance level of 0.05.

On average, the running time of the BRKGA was 96.50% shorter than that of the ALNS. On the group of instances that required the most computational effort ($m = 6, n = 200, l = 40$), the average running time of the ALNS was 14633.74 s, while that of the BRKGA was 2524.16 s. Indeed, the BRKGA reached the 3600 s time limit for only 207 of the 28800 runs (0.72% of the total).

A direct comparison of running times is not completely fair. The BRKGA is a parallel method utilizing four threads, while the ALNS is a sequential method whose one stopping criterion is a time limit. Notwithstanding, we performed an additional experiment with a single-threaded BRKGA and compared this to the ALNS using the same standards as in the previous comparison. For instances of the IPMTC-II set, the average running time of the sequential BRKGA was 83.79% shorter than that of the ALNS. Considering all instances, the parallel BRKGA as originally proposed was 78.39% faster than its sequential version on average, while keeping the same solution quality.

A last experiment was performed in order to verify if this particular BRKGA implementation is indeed an improvement over its original method, the RKGA. Considering both benchmark instances, the BRKGA found best solution values in 47.36% of the cases. The RKGA was able to find best solutions for 23.26% of

the instances, while both methods reached the same solution values for 29.36% of instances. The overall percentage distance between solution values, calculated as $\frac{RKGA-BRKGA}{BRKGA} \times 100$, varies between -1.57% and 17.76%, which indicates that the the superiority of the BRKGA over RKGA is substantial, while the RKGA presented a marginal superiority in a smaller number of cases.

6. Conclusion and future work

In this work, we proposed a new approach to the IPMTC. This is an $\mathcal{NP}$-hard problem, with several practical applications in the industrial sector. The new approach consists of an implementation of the parallel metaheuristic BRKGA hybridized with VND. The proposed method was analyzed using existing benchmark instances available in the literature, and compared with the current state-of-the-art. Considering the reported results and statistical analysis, we conclude that the proposed BRKGA is superior to the current state-of-the-art methods, as it produces high-quality solutions in a significantly shorter running time for nearly all benchmark instances available in the literature. When compared with the state-of-the-art mathematical model on a set of small instances, the BRKGA generated better or equal solution values for 99.86% of the instances. When compared with the state-of-the-art ALNS on a set of large instances, new best solutions were set for 93.89% of the instances. We credit the superior performance to the use of a metaheuristic that is suitable for permutation problems, and to the strengthening of the search intensification by means of operators for both IPMTC characteristics. By not neglecting any characteristic that is intrinsic to the problem, the proposed method achieves superior results and represents the new-state-of-the-art for the IPMTC. Future work will include addressing different versions of the IPMTC, such as environments with a single machine or non-identical machines, as well as tool-sharing and wearing scenarios.

Acknowledgments

This work was supported by National Counsel of Technological and Scientific Development (Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq) and Universidade Federal de Ouro Preto. The authors wish to convey their appreciation to Andreza Cristina Beezão, who kindly provided the instances used in the computational experiments.

References

Agnetis, A., Alfieri, A., Brandimarte, P., & Prinsecchi, P. (1997). Joint job/tool scheduling in a flexible manufacturing cell with no on-board tool magazine. *Computer Integrated Manufacturing Systems*, *10*, 61–68.

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

Allahverdi, A., Gupta, J., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, *27*, 219–239.

Allahverdi, A., Ng, C., Cheng, T. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European journal of operational research*, *187*, 985–1032.

Andrade, C. E., Ahmed, S., Nemhauser, G. L., & Shao, Y. (2017). A hybrid primal heuristic for finding feasible solutions to mixed integer programs. *European Journal of Operational Research*, *263*, 62 – 71.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, *6*, 154–160.

Beezão, A. C., Cordeau, J.-F., Laporte, G., & Yanasse, H. H. (2017). Scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, *257*, 834–844.

Beezão, A. C. (2019). *Mendeley Data - Results IPMTCII.* `http://dx.doi.org/10.17632/jv9tvby299.1`.

Berrada, M., & Stecke, K. E. (1986). A branch and bound approach for machine load balancing in flexible manufacturing systems. *Management Science*, *32*, 1316–1335.

Calmels, D. (2018). The job sequencing and tool switching problem: state-of-the-art literature review, classification, and trends. *International Journal of Production Research*, (pp. 1–21).

Chung, T., Gupta, J. N., Zhao, H., & Werner, F. (2019). Minimizing the makespan on two identical parallel machines with mold constraints. *Computers & Operations Research*, *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. *International Journal of Flexible Manufacturing Systems*, *6*, 33–54.

Fathi, Y., & Barnette, K. (2002). Heuristic procedures for the parallel machine problem with tool switches. *International Journal of Production Research*, *40*, 151–164.

Gökgür, B., Hnich, B., & Özpeynirci, S. (2018). Parallel machine scheduling with tool loading: a constraint programming approach. *International Journal of Production Research*, (pp. 1–17).

Gonçalves, J. F., & Resende, M. G. C. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, *17*, 487–525.

Graham, R., Lawler, E., Lenstra, J., & Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. Hammer, E. Johnson, & B. Korte (Eds.), *Discrete Optimization II* (pp. 287 – 326). Elsevier volume 5 of *Annals of Discrete Mathematics*.

Haddadi, S., Chenche, S., Cheraitia, M., & Guessoum, F. (2015). Polynomial-time local-improvement algorithm for consecutive block minimization. *Information Processing Letters*, *115*, 612–617.

Hertz, A., & Widmer, M. (1996). An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. *Discrete Applied Mathematics*, *65*, 319–345.

Koulamas, C. P. (1991). Total tool requirements in multi-level machining systems. *The International Journal of Production Research*, *29*, 417–437.

Kurz, M., & Askin, R. (2001). Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, *39*, 3747–3769.

Laporte, G., Salazar-Gonzalez, J. J., & Semet, F. (2004). Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions*, *36*, 37–45.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & operations research*, *24*, 1097–1100.

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

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 Journal of Operational Research*, *276*, 637 – 655.

Paiva, G. S., & Carvalho, M. A. M. (2017). Improved heuristic algorithms for the job sequencing and tool switching problem. *Computers & Operations Research*, *88*, 208–219.

Passmark (2018). *CPU Benchmarks*. `http://www.cpubenchmark.net/cpu_list.php` Accessed April 10, 2019.

Pinedo, M. (2018). *Scheduling: Theory, Algorithms, and Systems*. Springer.

Ramos, A. G., Silva, E., & Oliveira, J. F. (2018). A new load balance methodology for container loading problem in road transportation. *European Journal of Operational Research*, *266*, 1140 – 1152.

Shapiro, S. S., & Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, *52*, 591–611.

Spears, W. M., & De Jong, K. D. (1995). *On the virtues of parameterized uniform crossover*. Technical Report Naval Research Lab, Washington D.C.

Stecke, K. E. (1983). Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems. *Management Science*, *29*, 273–288.

Student (1908). The probable error of a mean. *Biometrika*, *6*, 1–25.

Tang, C. S., & Denardo, E. V. (1988). Models arising from a flexible manufacturing machine, part i: minimization of the number of tool switches. *Operations research*, *36*, 767–777.

Van Hop, N., & Nagarur, N. N. (2004). The scheduling problem of pcbs for multiple non-identical parallel machines. *European Journal of Operational Research*, *158*, 577–594.

Widmer, M. (1991). Job shop scheduling with tooling constraints: a tabu search approach. *Journal of the Operational Research Society*, *42*, 75–82.

Zeballos, L. (2010). A constraint programming approach to tool allocation and production scheduling in flexible manufacturing systems. *Robotics and Computer-Integrated Manufacturing*, *26*, 725–743.