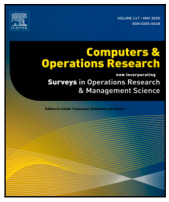
Computers & Operations Research 163 (2024) 106509

Contents lists available at ScienceDirect

Computers and Operations Research

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



Biased random-key genetic algorithm for the job sequencing and tool switching problem with non-identical parallel machines

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

Dataset link: https://github.com/larback/ssp-npm

Keywords:
Combinatorial optimization
Biased random-key genetic algorithm
Job sequencing and tool switching problem with non-identical parallel machines
Flexible manufacturing system

ABSTRACT

We address the job sequencing and tool switching problem associated with non-identical parallel machines – a variation of the well-known sequencing and switching problem (SSP) better adapted to reflect the challenges in modern production environments. The $\mathcal{NP}$-hard problem is approached by considering two isolated objective functions: the minimization of the makespan and the minimization of the total flow time. We present two versions of a parallel biased random-key genetic algorithm hybridized with tailored local search procedures that are organized using variable neighborhood descent. The proposed methods are compared with state-of-the-art methods by considering 640 benchmark instances from literature. For both objective functions considered, the proposed methods consistently outperform the compared methods. All known optimal values for both objectives are achieved, and a substantial gap is reported for all instance groups when compared with the best previously published solution values.

1. Introduction

The general complexity of the production management of a flexible manufacturing system (FMS) has inspired researchers to study several computational problems arising from such systems since the 1980s (Calmels, 2019). Among the several available FMS environments, the simplest comprises a single flexible machine connected by an automatic material handling system (Pinedo, 2012). A *flexible machine* can perform many different operations, such as cutting, drilling, and sanding (Zeballos, 2010), as long as the necessary tools, such as cut blades, drill bits, and sanders, have been loaded onto the tool *magazine*.

Usually, the magazine capacity can accommodate all the necessary tools for processing a single job. However, the number of tools required to process a sequence of jobs can exceed the magazine capacity of a flexible machine, thereby demanding tool switches between the sequential processing of two jobs. Tool switches represent approximately 30% of the fixed and variable costs in an FMS (Beezão et al., 2017), consuming more time than any other setup operation and directly affecting the cost of production (Van Hop and Nagarur, 2004).

Because tool switches are unavoidable in most real FMSs, their minimization has been studied in existing literature as the job sequencing and tool switching problem (SSP). Formally defined by Tang and Denardo (1988) and Bard (1988), the SSP comprises two main optimization problems: the sequencing of jobs on the flexible machine and the scheduling of the loading and unloading of tools onto the tool magazine. The first problem is $\mathcal{NP}$-hard (Crama et al., 1994), which determines the SSP complexity. However, given a fixed sequence of jobs to be processed, the second problem – the scheduling of tool switches – can be optimally solved in deterministic polynomial time using the *keep tool needed soonest* (KTNS) policy introduced in Tang and Denardo (1988).

Although its origin is in an industrial environment composed of a single flexible machine, the SSP appears in various industrial scenarios, such as microelectronic, metallurgic, steel, aerospace, and chemical industries (Shirazi and Frizelle, 2001; González et al., 2015; Soares and Carvalho, 2020), as well as scenarios involving multiple machines (Calmels, 2019). In fact, according to Calmels (2022), SSP approaches that consider environments comprising non-identical parallel machines are better adapted to the challenges in modern production environments.

In this study, we address the SSP in an industrial setting consisting of multiple non-identical parallel machines. This constitutes a variation of the SSP, named job sequencing and tool switching problem with non-identical parallel machines (SSP-NPM). We consider two objective functions separately: (i) the minimization of the *makespan* (C_{max}), i.e., the completion time of the most time-consuming machine (also called the *critical machine*); and (ii) the minimization of *total flow time* (TFT), i.e., the sum of the completion times of all jobs. The main contribution of this study is the development of a biased random-key genetic

* 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, leo@larback.com.br (L.C.R. Soares), mamc@ufop.edu.br (M.A.M. Carvalho).

https://doi.org/10.1016/j.cor.2023.106509
Received 11 July 2023; Received in revised form 31 October 2023; Accepted 7 December 2023
Available online 13 December 2023

Table 1
SSP-NPM Instance.

Jobs	1	2	3	4	5	6
Tools	1	2	1	3	3	2
	3	4	2	5	5	3
	7	8	5	7	9	4
	9			8		6
p_{1j}	∞	5	8	∞	2	∞
p_{2j}	1	4	7	5	1	9

algorithm (BRKGA) hybridized with local search procedures organized using a variable neighborhood descent (VND) method to solve the SSP-NPM, which outperforms existing algorithms. Furthermore, we publish new best results for all benchmark instances available in the literature for the two objective functions considered. We expect that the new results will challenge and inspire researchers to further investigate the problem, thereby increasing the number of directly related publications and strengthening the literature on this important practical problem.

The remainder of this paper is organized as follows. The SSP-NPM is formally described in Section 2. Then, Section 3 presents the related works. Next, Section 4 presents the details of the proposed methods, 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 non-identical parallel machines $M = \{1, \ldots, m\}$, with each machine containing a tool magazine with the capacity for C_i $(i \in M)$ tools loaded simultaneously; a set of jobs $J = \{1, \ldots, n\}$, each with a processing time indicated by p_{ij} $(i \in M, j \in J)$; a set of tools $T = \{1, \ldots, l\}$; a subset of tools T_j required to process each job j $(j \in J)$; and the time $\bar{p}_i$ $(i \in M)$ required to perform tool switches in each machine. The SSP-NPM entails the assignment of jobs to machines, their sequencing on each machine, and the resulting tool switching plan for each machine, such that the considered objective function is optimized.

As with the SSP, jobs have no release or due times, are not preemptive, and have no precedence order. Notably, the non-identical parallel machines are unrelated and have different magazine capacities, which implies an eligibility constraint, i.e., a job can only be processed by a machine if the machine magazine capacity is equal to or greater than the number of tools required by the job ($C_i \geq |T_j|$). At least one machine must contain a magazine with sufficient capacity to hold all the tools required to process any single job. Each machine has a complete set of tools, which can be loaded in any position available in the magazine. Tool switches occasioned by damaged or broken tools are not considered. The setup time required for the initial configuration of the machine is also not considered.

Table 1 shows an instance for the SSP-NPM that considers $n = 6$, $m = 2$, $l = 9$, $C_1 = 3$, $C_2 = 4$, $\bar{p}_1 = 3$, and $\bar{p}_2 = 2$. If a machine is ineligible for a given job, the processing time for that job on that machine is represented by ∞.

Regarding the objective function, the current state-of-the-art approach to SSP-NPM (Calmels, 2022) considers separately the minimization of the total number of tool switches (TS), minimization of the makespan (C_{max}), and minimization of the total flow time (TFT). However, the paper highlights that, in environments composed of multiple machines, the minimization of makespan and total flow time are metrics with more practical applications and academic relevance.

In fact, when exclusively considering the minimization of the total number of tool switches in an environment composed of non-identical parallel machines with magazines of different capacities, the solution structure tends to underutilize machines with magazines of smaller capacities, thus resulting in prohibitively large production times. This solution structure is also caused by the fact that the SSP-NPM does not consider the initial loading of tools as tool switches, inducing the use of machines with smaller magazine capacities only for small subsets of jobs that do not result in tool switches. This indicates that the minimization of tool switches is more suitable for single-machine environments. Therefore, in this study, the SSP-NPM is approached by considering only the minimizations of the makespan and the total flow time as the objective functions, because these reflect practical applications. Section 5.4.4 includes a discussion concerning the three different objective functions, including a graphical representation of the solution structure when TS is considered as the objective function.

A solution to an SSP-NPM instance is provided by the sequencing of jobs for each machine, the tool switching plan for each machine, and its objective value. 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 machine i in processing order. An entry $r^i_{tj} = 1$ indicates that tool t is loaded onto 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 using Eq. (1), as proposed by Crama et al. (1994). This equation counts the number of inversions from 0 to 1, representing tool insertions into the magazine.

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

The completion time of a machine is given by the sum of the job processing times 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 makespan is calculated according to Eq. (2).

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

The total flow time is given by Eq. (3). Let the variable f_j be the completion time of job j (which includes the time spent on tool switches); the equation sums up the completion times of all jobs.

$$\text{TFT} = \left\{ \sum_{j \in J} f_j \right\} \tag{3}$$

Although the objectives are conflicting (Calmels, 2022), it is expected that solutions generated while minimizing the total flow time may have good makespan values, because the makespan is given by the longer completion time between jobs. However, solutions generated while minimizing the makespan may not present good values of total flow time, given that solutions generated with this objective tend to place less importance on *non-critical machines*. By considering the two objectives separately, this study provides a better overview of the production throughput resulting from the scheduling obtained, providing a more significant basis for decision-making in practical industrial applications.

3. Related works

Although general studies related to scheduling jobs on parallel machines date back to the late 1950s, the first mention close to the definition of the SSP with multiple machines only appeared in the early 1980s, when (Stecke, 1983) listed five production planning problems in an FMS that aimed to maximize production and reduce costs.

The majority of articles approaching the SSP for multiple machines consider an environment composed of identical parallel machines, a variation of the problem known as the identical parallel machine problem with tooling constraints (Soares and Carvalho, 2020). However, to the best of our knowledge, only a few papers have been published that consider the SSP for non-identical parallel machines, making the SSP-NPM a novel contribution to the SSP literature (Calmels et al., 2019). Next, we present the specific SSP-NPM approaches. For an overview of SSP, including variations with sequence-dependent setup

times, multiple machines, multiobjective approaches, and similar problems, see Allahverdi et al. (1999, 2008), Allahverdi (2015), Calmels et al. (2019).

Özpeynirci et al. (2016) and Gökgür et al. (2018) address the SSP-NPM aiming to minimize the makespan. There are no machine eligibility constraints; however, the job processing times vary for each machine, and a limited number of tool copies are considered. Özpeynirci et al. (2016) presents two mathematical models and a tabu search for the problem. The computational experiments adopted a set of 70 instances proposed in the same study. With a one-hour time limit, the models proved inefficient for the largest instances. The tabu search provided near-optimal solutions for small instances and improved the model's upper bounds for the largest instance. Gökgür et al. (2018) proceeds with the study of the same problem by taking a constraint programming approach. The proposed formulation performed better than previous models and tabu search implementations for large instances.

Calmels et al. (2019) and Calmels (2022) address the SSP-NPM considering the same environment described in Section 2. The authors consider three objective functions: (i) minimization of the total number of tool switches; (ii) minimization of the makespan; and (iii) minimization of the total flow time. Calmels et al. (2019) presents three constructive heuristics for the problem. The computational experiments used a previous benchmark instance proposed by Beezão et al. (2017) for the problem of identical parallel machines with tooling constraints, adapted for non-identical parallel machines.

Calmels (2022) presents a mixed integer linear programming (MILP) model, three construction heuristics, and three different implementations of the iterated local search (ILS) metaheuristic. The difference between the ILS implementations lies in the use of three objective-specific perturbation strategies. The computational experiments considered 640 newly proposed instances, which are described in Section 5.1. To minimize tool switches and total flow time, the MILP model reported optimal solutions for all 40 smallest instances, with ten jobs. Considering makespan minimization, the model reported optimal solutions for 38 of the 40 smallest instances. For instances with up to 20 jobs, the ILS solutions were compared with each other and with the optimal solutions and upper bounds reported by MILP. Considering the minimization of tool switches, the MILP model significantly outperformed the ILS methods.

Regarding the time-related objectives, i.e., makespan and total flow time, the values of the solutions reported by at least one of the ILS implementations were better than those reported by MILP. All optimal values reported for the makespan and 38 of 40 optimal solutions reported for the total flow time were matched. On average, the ILS improved the upper bounds values reported by the MILP for makespan and total flow time by 0.40% and 2.02%, respectively. For large instances, the ILS methods were compared only with each other, emphasizing the impact of the different perturbation methods considered.

In a recent study, Cura (2023) addressed the SSP-NPM in the same scenario considered by Calmels (2022). The author presents two versions of a genetic algorithm (GA) hybridized with local search. The GA versions differ in the policy used to determine which tool will be removed from the magazine. The first version uses the well-known KTNS policy, while the second chooses tools randomly. The computational experiments considered the same objectives and instances proposed by Calmels (2022). Considering the three objectives, the proposed GAs presented better or equal results for all except two subsets of instances. For the two smallest subsets of instances, the optimal results reported by the MILP (Calmels, 2022) for the minimization of total flow time were not matched.

By presenting the best-known results for the only instance benchmark available in the literature on the SSP-NPM, Calmels (2022) and Cura (2023) represent the current state-of-the-art for the problem. In the experiments reported in Section 5, we consider the same instances and compare the results of our methods with the best-known results for makespan and total flow time generated either by the MILP (Calmels, 2022) or the different versions of GA (Cura, 2023).

4. Methods

To approach the SSP-NPM, we present two versions of the evolutionary metaheuristic BRKGA (Gonçalves and Resende, 2011) hybridized with three local search procedures organized via a VND (Mladenović and Hansen, 1997). Both versions differ in terms of the objective function, hybridization strategy, and neighborhoods considered. We chose the BRKGA because of its suitability for permutation problems, such as SSP-NPM, and its recent good performances on relevant combinatorial problems (Andrade et al., 2017; Ramos et al., 2018; Oliveira et al., 2019; Kummer et al., 2020). The hybridization process is similar to that proposed in Soares and Carvalho (2020, 2022) for related problems. The rationale is to include intrinsic characteristics of the SSP-NPM in the evolutionary process of BRKGA. In the following sections, we describe the proposed implementations, the hybridization component, and the differences between both versions.

4.1. Biased random-key genetic algorithm

Inspired by the random-key genetic algorithm (Bean, 1994), the BRKGA follows a common framework of the genetic algorithm: a *population* of pop randomly generated *individuals* (potential solutions) evolves over some *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 *elite*. The remaining $pop - pop_e$ individuals are considered *non-elite*. The population of the next generation is formed from that of the current generation through the application of selection, reproduction, and mutation operators.

Specific to the BRKGA, each individual is encoded as a vector of randomly generated *keys* in the continuous interval [0, 1]. In the evolutionary process, elite individuals are copied from the current generation to the next, creating a new population. New pop_m individuals called *mutants* are randomly generated and introduced into the new population. The remaining portion of the new population is formed by $pop - pop_e - pop_m$ *offspring* generated by combining two parent individuals selected randomly from the current generation – one from the elite group and the other from the non-elite group – using the *parametrized uniform crossover* (Spears and De Jong, 1995), in which the ith key of an offspring is copied from one of the parents, with a higher probability ρ of inheriting from the elite parent.

Fig. 1 illustrates the parametrized uniform crossover considering $\rho = 0.75$. The parents are chosen randomly, one from the elite set and the other from the non-elite set. For each gene, a random number is generated in the continuous interval [0, 1]. If its value is lower than ρ, the offspring receives the gene of the elite parent. Otherwise, the offspring receives the gene of the non-elite parent. The use of the parametrized uniform crossover adds another layer of elitism to the BRKGA evolutionary process. According to the value set for ρ, the method may be more or less biased to inherit from the elite individual.

The only problem-dependent component of the BRKGA is the *decoder*, an algorithm that transforms a vector of random keys into a solution for a particular optimization problem. Both encoding and decoding are explained next.

4.1.1. Encoding and decoding

To represent an SSP-NPM solution, keys are generated in the interval $[0, |M_j|) \in \mathbb{R}$, where M_j is the set of machines eligible to process job j. The number n of jobs defines the number of keys per individual. The rationale behind this encoding is that the random-key integer part represents an index in set M_j, and is used to select one of the machines eligible to process job j. The decimal part of the key is used to determine the job sequencing in a specific machine.

Fig. 2 illustrates the encoding of an individual for an SSP-NPM instance containing three machines and eight jobs: $M_0 = \{1, 2, 3\}$, $M_1 = \{1\}$, $M_2 = \{1, 2, 3\}$, $M_3 = \{1, 3\}$, $M_4 = \{1, 3\}$, $M_5 = \{1, 3\}$, $M_6 = \{1, 2, 3\}$, and $M_7 = \{1\}$.

Fig. 1. Example of parametrized uniform crossover with $\rho = 0.75$.

Index	0	1	2	3	4	5	6	7
Genes	1.315	0.963	1.001	1.982	0.754	1.936	1.541	0.127

Fig. 2. Encoding of an individual for instance with three machines and eight jobs.

Original index	7	4	1	2	0	6	5	3
Genes	0.127	0.754	0.963	1.001	1.315	1.541	1.936	1.982

Fig. 3. Sorted individual on the decode step.

An individual is decoded into a valid SSP-NPM solution by sorting its keys in non-decreasing order while keeping track of their original indexes in the vector. The original index of each gene $[0, \ldots, n-1]$ indicates the job it represents. The integer part of the key indicates the index in the set M_j that contains the machine to which the job will be assigned – the same index on different sets M_j may represent different machines, according to the eligibility constraint. The sorted keys determine the job schedule on each machine. For example, the individual from Fig. 2 is sorted in Fig. 3. Thus, this example corresponds to the assignment of jobs 7, 4, and 1 to machine 1 ($M_7(0) = 1, M_4(0) = 1, M_1(0) = 1$); jobs 2, 0, and 6 to machine 2 ($M_2(1) = 2, M_0(1) = 2, M_6(1) = 2$); and jobs 5 and 3 to machine 3 ($M_5(1) = 3, M_3(1) = 3$), in that order.

The fitness of an individual is calculated according to the objective function considered for the problem after the decoding step. In the version named BRKGA$_{Cmax}$, fitness is given by the makespan calculated according to Eq. (2). In the version named $BRGKA_{TFT}$, fitness is given by the total flow time calculated according to Eq. (3).

4.1.2. Hybridization

To incorporate the characteristics of the SSP-NPM in the BRKGA evolutionary process, we hybridized it with local search procedures tailored for the objective functions considered in this study. After decoding an individual as an SSP-NPM solution, local search procedures are applied to it as a specific component of search intensification. For elite individuals, local search procedures are organized in a VND. For non-elite individuals, they are applied linearly. The different hybridization strategies are defined in Section 5.2. Eventual changes in the solution are replicated in the encoding of the individual, and its fitness value is updated.

Algorithm 1 presents the pseudocode for the proposed hybrid BRKGA. The input consists of the number of jobs (n), number of machines (m), the set of machines (M_j) eligible to process each job j, population size (pop), size of the elite group (pop_e), number of mutant individuals (pop_m), and the probability ρ of inheriting the genes of the elite parent. First, the initial population is randomly generated (line 1). Then, the main loop (lines 2–22) evolves the population while the stopping criteria are not met. Each new individual is decoded into an SSP-NPM solution (line 3), which is then evaluated (line 4). The VND is performed on elite individuals (line 5). The local search procedures (job insertion, job exchange, and 1-block grouping) are applied linearly once to the remaining individuals (line 6). All solutions are encoded back and their fitness values are updated (line 7). The individuals are then sorted, and the new elite set is selected (lines 8 and 9). The elite set is copied to the next generation, and new mutant individuals are introduced (lines 10 and 11). The inner loop (lines 12–21) generates the remaining individuals by randomly selecting two parents, one from the elite set (line 13) and the other from the non-elite set (line 14). Then, the offspring inherits each of its genes (lines 15–21) either from the elite parent (line 18), with probability $1 - \rho$ (line 16), or from the non-elite parent (line 20). Each newly generated individual is then added to the next generation (line 21). When complete, the next generation replaces the previous one (line 22), and the process restarts until the stopping criterion is met.

Algorithm 1: Hybrid biased random-key genetic algorithm

```
   input : n, m, M_j, pop, pop_e, pop_m, ρ
1  Generate population Π with pop vectors of n random keys
     chosen from [0, |M_j|);
2  while stop criterion is not met do
3      Decode each new individual;
4      Evaluate the fitness for each decoded individual in Π;
5      Apply the VND on the pop_e first solutions;
6      Apply the local search procedures on the pop − pop_e
         individuals;
7      Encode each solution as individuals and update their
         fitnesses;
8      Sort individuals in non-decreasing fitness order;
9      Group the first pop_e individuals in Π_e;
10     Initialize the population of the next generation Π+ ← Π_e;
11     Generate a set of pop_m mutants and add it to Π+;
12     for i ← 1 to pop−pop_e−pop_m do
13         Select a random individual a from the elite group;
14         Select a random individual b from the non-elite group;
15         for j ← 1 to n do
16             Throw a biased coin with probability ρ of heads;
17             if heads then c[j] ← a[j]; else c[j] ← b[j];
18             Π+ ← Π+ ∪ {c};
19     Π ← Π+;
```

In order to speed up the BRKGA execution, the decoding, evaluation and local search steps are parallelized (lines 3–7). Therefore, different individuals can be simultaneously decoded, evaluated, improved by local search and encoded. We define the maximum number of threads used by BRKGA in Section 5.2. Fig. 4 illustrates the parallelization of these steps, considering a hypothetical scenario with three available threads.

4.1.3. Stopping criteria

The maximum number of generations is used as the stopping criterion for both versions of the BRKGA, whose value is defined in the preliminary experiments described in Section 5.2. Additionally, aiming for a reasonable comparison with the results reported by Calmels (2022), a trigger was added that interrupts the BRKGA operation after

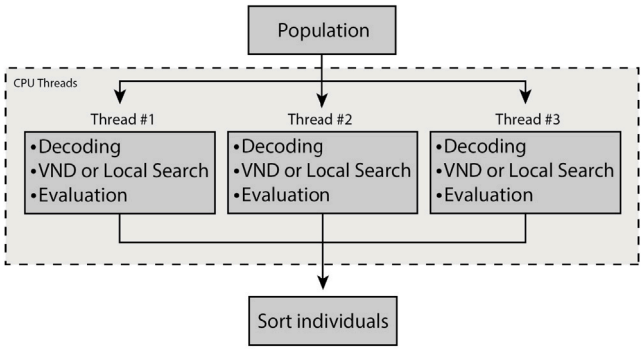


Fig. 4. Parallel scheme for decoding step and local search applications.

3600 s, according to the original experiment. The original computational environment included a CPU with (Passmark, 2023) single-thread score of 2.766, which is 79.50% higher than that adopted in this study. Nonetheless, we credit such time limit to practical limitations of industry, rather than computational limitations.

4.2. Variable neighborhood descent

The metaheuristic VND, introduced by Mladenović and Hansen (1997), consists of systematic alterations of neighborhood structures while exploring the solution space to find a local optimum shared by the different neighborhoods. Our implementation considers three local search methods: job insertion, job exchange, and 1-block grouping. The following sections describe the details of these local search methods.

Algorithm 2 presents the pseudocode for the proposed VND. A feasible solution s is provided as an input parameter. The loop (lines 2–13) 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 solution is evaluated and compared with the previous one (line 9). If the current value is better, then s is updated and the exploration restarts from the first neighborhood (lines 10 and 11). Otherwise, the VND proceeds to the next neighborhood (line 13). Finally, when exploring all neighborhoods does not improve the current solution, s is a local optimum common to all neighborhoods and is returned.

Algorithm 2: Variable neighborhood descent

```
   input : solution s
1  k = 1;
2  while k ≠ 4 do
3      if k = 1 then
4          s' ← JobInsertion(s);
5      if k = 2 then
6          s' ← JobExchange(s);
7      if k = 3 then
8          s' ← OneBlocksGrouping(s);
9      if evaluation function (s') < evaluation function (s) then
10         s ← s';
11         k ← 1;
12     else
13         k ← k + 1;
14 return s;
```

The order of application of local search procedures was defined according to the main characteristics of each and was common to both versions of BRKGA. Initially, job insertion is applied to balance the completion times among all machines. Balanced solutions tend to have less room for improvements by simple insertions. Thus, job exchange is applied to improve the solution through cross-insertions. After focusing on solving the job scheduling problem for non-identical parallel machines, we shift the focus to the SSP solution for each machine. To reduce the required setup time and, eventually, free space for new improvements by insertions, 1-block grouping is applied. The preliminary experiments described in Section 5.2 defined the strategy for applying VND in the BRKGA hybridization process. Next, we describe and illustrate each local search method.

4.2.1. Job insertion

To avoid machine idleness, this local search attempts to balance the processing times among machines, moving jobs assigned to machines with longer completion times to less time-consuming machines. To this end, the machines are sorted in non-decreasing order of processing time. The jobs eligible for the less time-consuming machines are then randomly selected one at a time from the critical machine. The selected job is removed from the critical machine and assigned to a less time-consuming machine in a position that results in the least number of additional tool switches. The neighbor solution is evaluated, and if there is an improvement in the evaluation value, the machines are sorted again, and the process restarts from this new solution. After all the jobs from the critical machines have been considered and no improvement in the solution value is possible, the local search ends in BRKGA_{Cmax}. In BRKGA_{TFT}, the previous second most time-consuming machine is then regarded as the critical one, and the process restarts. The local search ends when no insertion movement improves the current solution value.

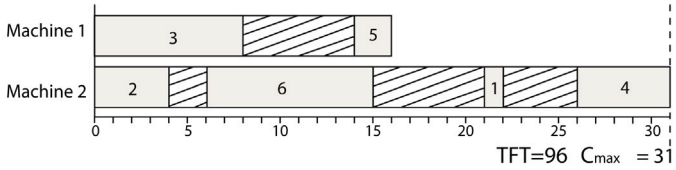
Fig. 5 illustrates a hypothetical job insertion considering a valid solution for the SSP-NPM instance presented in Table 1. This instance will be used to illustrate each local search method. Rectangles represent jobs, with widths proportional to processing times, whereas the hatched area represents the required setup time. In (a), we obtained a feasible solution with TFT = 96 and C_{max} = 31. A random job is selected from the critical machine (b). Next, all positions for insertion on machine 1 are evaluated (c), and the position that yields the least number of tool switches is selected (d), generating a neighbor solution with TFT = 101 and C_{max} = 27. This new solution is accepted by BRKGA_{Cmax} because it results in a lower makespan value. However, it is rejected by BRKGA_{TFT} because it increases the total flow time.

4.2.2. Job exchange

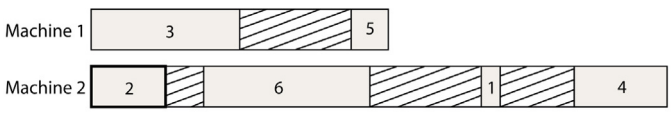
SSP-NPM solutions with balanced machine processing times tend to have less space for simple insertions to improve the objective functions considered in this study. Thus, this local search performs crossed insertions by exchanging two jobs assigned to different machines. This movement can improve the solution given the different job processing times in each machine, or because it results in a smaller number of tool switches. To this end, the machines are sorted in non-decreasing order of processing time. For each job assigned to the critical machine and eligible for the less time-consuming machine, each job currently assigned to the less time-consuming machine is considered for exchange. For both machines, the jobs are randomly selected.

To reduce the neighborhood size, we consider a minimum compatibility policy proposed by Fathi and Barnette (2002), where both machines involved in the exchange must have at least 50% of tools in common, considering their assigned jobs, except for the two jobs considered in the exchange. If the machines are compatible, the jobs are exchanged and the new solution is evaluated. If the solution value improves, the solution is updated and the process restarts. Otherwise, the exchange is undone and the next job is considered for exchange. This local search ends when no exchange movement improves the current solution.

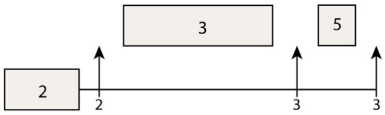
Fig. 6 illustrates a hypothetical job exchange movement. In (a), we have a feasible solution with TFT = 109 and C_{max} = 36. In (b), we demonstrate the random selection of jobs. In this example, job 3 on the critical machine and job 5 on the less time-consuming machine are selected. These jobs are eligible for destination machines and have the minimum tool compatibility. Therefore, the exchange is performed, generating a neighbor solution with TFT = 94 and C_{max} = 26, as shown in (c).



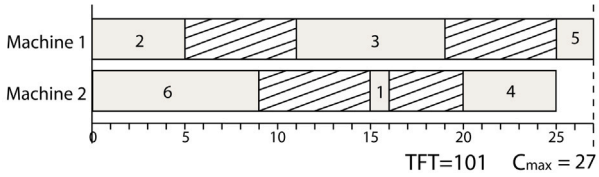
(a) Initial solution.



(b) Random job selection of the critical machine.

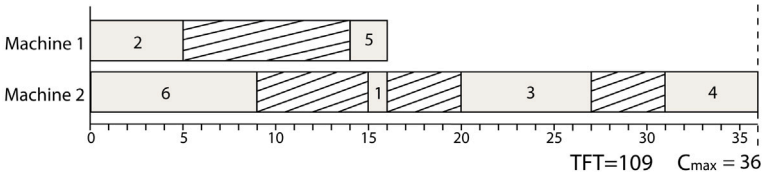


(c) Insertion in the best position.

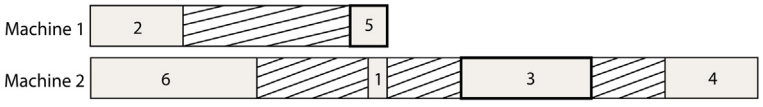


(d) New solution.

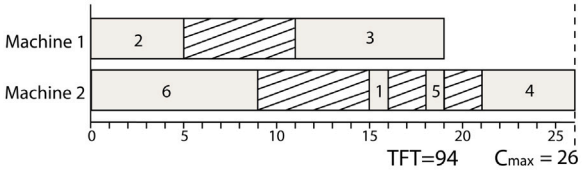
Fig. 5. Job-insertion local search.



(a) Initial solution.



(b) Random job selection.



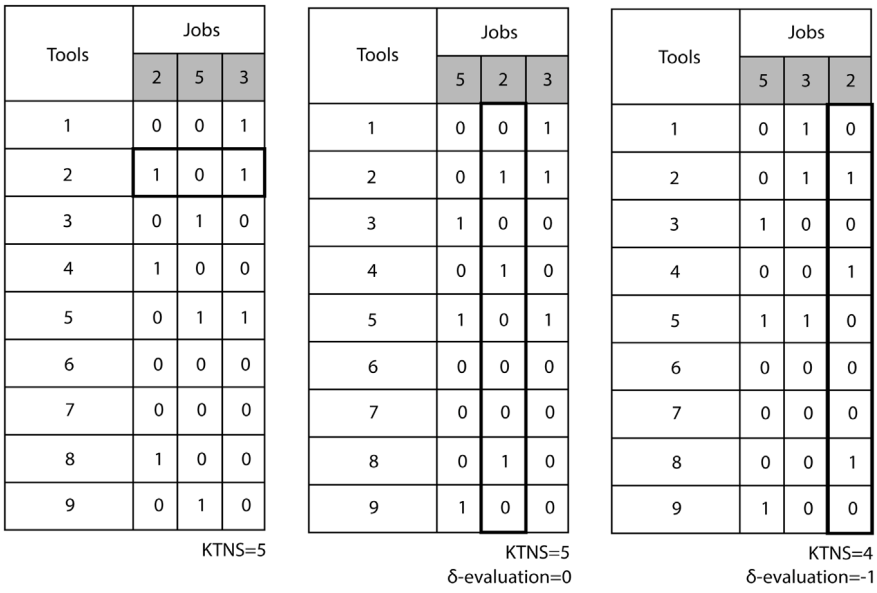
(c) New solution.

Fig. 6. Job-exchange local search.

4.2.3. 1-Block grouping

As mentioned in Section 2, an SSP-NPM solution is determined by sequencing jobs for each machine and by the evaluation value. The tool switches on each machine, responsible for the machine setup time, can be represented by a binary matrix $R^i = \{r^i_{tj}\}$, where $r^i_{tj} = 1$ indicates that tool t is loaded on machine i during the processing of job j, and $r^i_{tj} = 0$ otherwise. A 1-block is a contiguous sequence of nonzero elements in the same row in a binary matrix. For the SSP-NPM, two or more 1-blocks in the same row indicate that one specific tool is unloaded and loaded again in the magazine at a later production stage. This also indicates a point for possible improvement in the machine setup time. Therefore, this local search, first introduced by Paiva and Carvalho (2017), attempts to reduce the number of tool switches on a machine by grouping the 1-blocks in each row of matrix R^i.

To this end, each machine in the SSP-NPM solution is considered an SSP instance. The tool matrix is analyzed row by row in random order. If the row has two or more 1-blocks, the local search moves the columns of the first 1-block one by one to the left or right of the second 1-block to group them. Each movement is analyzed, and all movements that decrease the machine setup time are accepted. If both movements tie in solution value, the last evaluated movement is maintained. To avoid a complete evaluation of movements that cannot decrease the machine setup time, we use the δ-evaluation proposed by Haddadi et al. (2015). This fast evaluation procedure verifies the impact on the number of 1-blocks after a single column is moved from its original position or when two columns exchange positions. Only movements with zero or negative δ-evaluation values can decrease the machine setup time (Soares and Carvalho, 2020) and are reevaluated using the KTNS policy. This local search is limited to the critical machine when



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

Fig. 7. 1-block-grouping local search.

applied by the BRKGA$_{Cmax}$. Conversely, all machines are individually considered in the BRKGA$_{TFT}$.

Fig. 7 illustrates a hypothetical grouping of 1-blocks. In (a), we have the machine tool matrix, the number of tool switches (KTNS), 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) preserves 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 other hand, moving it to the right of the second 1-block (c) decreases the number of 1-blocks and tool switches. Therefore, the last movement is maintained.

5. Computational experiments

Computational experiments were performed to configure the two versions of BRKGA and compare them with state-of-the-art methods. The following sections describe benchmark instances and report the experimental results.

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 7 operating system and an individual thread score of 1.541 (Passmark, 2023). The BRKGA was implemented in C++ and compiled using GCC 10.1.0 and options -O3 and *-march = native*. The (OpenMP, 2019) shared-memory parallel programming API version 5.0 was used to parallelize the BRKGA decoding and local search steps. Detailed results for all experiments described below as well as the source code of both BRKGA versions are available at https://github.com/larback/ssp-npm.

5.1. Instances

As provided by Calmels (2022), the only benchmark instance available for SSP-NPM is composed of 640 instances separated into two sets, SSP-NPM-I and SSP-NPM-II, containing 160 and 480 instances, respectively. Table 2 lists the characteristics of each set. The column headings represent the number of instances (#), jobs (n), machines (m), tools (l), and magazine capacity (C_i) of each machine i.

Table 2
Characteristics of the instances.

set	#	n	m	l	C_i
SSP-NPM-I	80	{10, 15}	2	{10, 15}	[5, 7]
	80	{15, 20}	3	{15, 20}	[7, 10, 13]
SSP-NPM-II	320	{40, 60}	4	{60, 120}	[25, 30, 35, 40]
	160	{80, 120}	6	{120}	[45, 50, 55, 60, 65, 70]

Table 3
Values considered and selected by irace to define the parameters of BRKGA$_{Cmax}$.

Parameter	Options
Population size (pop)	$\{n, 2\times n, 5\times n, \mathbf{10\times n}\}$
Elite population portion (pop_e)	{10%, 15%, **20%**, 25%, 30%}
Mutant population portion (pop_m)	{5%, 10%, 15%, 20%, **25%**, 30%}
Bias towards elite parents (ρ)	{60%, 65%, 70%, **75%**, 80%, 85%, 90%, 95%}

5.2. Parameter tuning

Experiments were conducted to determine the final configuration of each BRKGA version. The parameters of the metaheuristic were tuned using the *irace* offline method of automatic configuration for optimization algorithms (López-Ibáñez et al., 2016). Given a set of test instances and the possible values for each parameter, *irace* determines the appropriate combination. The set of instances used in these experiments comprised 10% of the available instances that were randomly selected.

Tables 3 and 4 present the values considered and selected (bold values) by irace. Although the upper limit for the population size was selected for both BRKGA versions, larger values were not considered for this parameter to avoid an excessively long running time. Despite the different objectives, the values selected by irace for both BRKGA versions were very close. The main difference lies in the value selected for the mutant portion of the population, indicating that $BRKGA_{Cmax}$ requires greater population diversification.

Additionally, the hybridization strategy in the decoding step and the BRKGA maximum number of generations were defined in preliminary experiments. To avoid premature convergence and long running times, distinct strategies were used for individuals belonging to elite and non-elite sets. Considering the elite set, VND was applied to 5%

Table 4
Values considered and selected by irace to define the parameters of $BRKGA_{TFT}$.

Parameter	Options
Population size (pop)	$\{n, 2\times n, 5\times n, \mathbf{10\times n}\}$
Elite population portion (pop_e)	$\{10\%, 15\%, 20\%, \mathbf{25\%}, 30\%\}$
Mutant population portion (pop_m)	$\{5\%, \mathbf{10\%}, 15\%, 20\%, 25\%, 30\%\}$
Bias towards elite parents (ρ)	$\{60\%, 65\%, \mathbf{70\%}, 75\%, 80\%, 85\%, 90\%, 95\%\}$

of its individuals by $BRKGA_{Cmax}$ and to 1% of its individuals by $BRKGA_{TFT}$. In both versions, the individuals on which VND is applied are marked to avoid reapplication in future generations, as the elitism mechanism present in the method does not alter the individuals. Considering the non-elite individuals, $BRKGA_{Cmax}$ apply the local search procedures linearly, and only once, to 50% of randomly selected individuals. The $BRKGA_{TFT}$ modulates the proportion of randomly selected individuals in accordance with the population size. Specifically, in populations comprising 600 or more individuals, local search procedures are applied in 2% of the individuals. In populations consisting of 400 individuals, this proportion is increased to 5%, whereas in populations with fewer than 400 individuals, the local search procedures are applied to 10% of the individuals. The maximum number of BRKGA generations in both versions was set to 1000 in an attempt to balance the running time and solution quality. Finally, the BRKGA decoding step and local search applications were parallelized. To avoid any possible difficulties in reproducing the reported experiments, the maximum number of threads used by BRKGA was limited to four. However, we believe this number can be easily increased in several other computational environments.

5.3. Preliminary experiments

A set of preliminary experiments was conducted in order to analyze the individual BRKGA components against the hybrid versions. This experiment considered both versions of 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 analyzes.

Considering the hybrid BRKGA as the reference and the SSP-NPM I set, from a makespan perspective, the original BRKGA reported an average percentage distance of 0.79%, ranging between 0.00% and 9.10%. The VND reported an average percentage distance of 20.01%, ranging between 0.00% and 60.00%. Regarding the total flow time, the original BRKGA reported an average percentage distance of 0.14%, ranging between −1.02% and 2.74%. The VND reported an average percentage distance of 11.95%, ranging between 0.00% and 29.55%.

For the SSP-NPM II set, regarding the makespan, the original BRKGA reported an average percentage distance of 6.54%, ranging between −4.86% and 17.69%. The VND reported an average percentage distance of 11.68%, ranging between 0.00% and 37.82%. Concerning the total flow time, the original BRKGA reported an average percentage distance of −0.49%, ranging between −13.18% and 7.58%. The VND reported an average percentage distance of 26.93%, ranging between 3.60% and 79.01%.

As can be observed in the percentage distance average values, the original BRKGA outperformed the hybrid version for TFT on some instances, 2 in SSP-NPM-I and 163 in SSP-NPM-II. It also happened concerning makespan on six instances. This is owed to the limit on maximum running time. As the original BRKGA does not include local searching, it can perform more iterations of the evolutionary process than the hybrid versions, particularly in large instances, e.g., those with six machines. In 169 cases, this particularity led to the best solution values, which could be matched or improved by the hybrid algorithms in longer running times.

The original BRKGA presented an erratic behavior concerning solution values for TFT, as it generated solution values with large deviation for the same instances, including a single case with percentage distance of −13.18% to the hybrid version solution value, which highly influenced the average values. Overall, the average percentage distance between the solution values generated by the GAs substantially favors the hybrid versions. Additionally, the sinergy of components in the hybrid BRKGA generated a considerably more robust and reliable algorithm. Therefore, we chose to consider it in the comparison with the state-of-the-art. Detailed results of the preliminary experiment are provided in the Supplementary Material.

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

After defining the final configuration of both BRKGA versions, a series of computational experiments were conducted to assess the quality of the solutions reported by the proposed methods against the results published by state-of-the-art methods (Calmels, 2022; Cura, 2023). When reporting the average results obtained, Calmels (2022) and Cura (2023) rounded them to integer values. The values reported in this section follow the same pattern. However, given the high competitiveness between methods proposed for similar problems, we recommend that future studies should use values with greater precision. The detailed results of each experiment can be found in the Supplementary Material.

As mentioned in Section 3, Calmels (2022) presented a MILP model and Cura (2023) presented two versions of a hybrid genetic algorithm applied to the SSP-NPM solution. We use as reference values the best results reported by any of these methods for the two objective functions considered in this study.

5.4.1. Comparison on SSP-NPM-I set

Table 5 presents the reference values and the average of the best values reported considering ten independent BRKGA runs for each instance of the SSP-NPM-I set. It shows the number of machines (m), jobs (n), tools (l), and average reference values (ARV) of makespan (C_{max}) and total flow time (TFT). For each BRKGA version, we present the average values of makespan (C_{max}), total flow time (TFT), running time (T) in seconds, and the percentage difference (gap(%)) between the best average values reported by BRKGA and the reference values for each objective function, calculated as $100\times\frac{\text{BRKGA - ARV}}{\text{ARV}}$. Bold values indicate the best results for each group of instances.

Concerning the makespan and SSP-NPM-I set, $BRKGA_{Cmax}$ reported average results that were better than or equal to those reported by the state-of-the-art methods for all groups of instances, presenting an average gap of −0.91%. As mentioned in Section 3, the model proposed by Calmels (2022) reported optimal makespan values for 38 of the 40 instances containing 10 jobs. $BRKGA_{Cmax}$ reported solutions with equal values for all instances.

On average, the convergence of $BRKGA_{Cmax}$ occurs after 11.77 generations. The initial solution was improved by 33.19%, and the average running time was only 8.35 s. Among the local search procedures that constitute the hybridization component, the local search that contributed the most to the quality of the final solution was job insertion, accounting for 54.03% of the improvements. Job exchange and 1-block grouping accounted for 32.60% and 13.37% of the improvements, respectively. The average standard deviation reported for all instances in this set was only 0.007, demonstrating the consistency of the method in generating solutions with small variations over independent runs. The additional stopping criterion regarding the maximum running time was not deployed for any instance in this set.

Regarding the total flow time, $BRKGA_{TFT}$ reported new best or equal average solution values for all groups of instances, with an average gap of −0.45% compared with the state-of-the-art methods. For all ten-job instances for which the optimal solution values are known, $BRKGA_{TFT}$ reported solutions with the same values. On average, the convergence of $BRKGA_{TFT}$ occurs after 81.81 generations; the initial

Table 5
Results for the SSP-NPM-I set of instances.

m	n	l	ARV		$BRKGA_{Cmax}$					$BRKGA_{TFT}$				
			C_{max}	TFT	C_{max}	TFT	T	gap_{Cmax}(%)	gap_{TFT}(%)	C_{max}	TFT	T	gap_{Cmax}(%)	gap_{TFT}(%)
2	10	10	**29**	**129**	**29**	146	1.75	0.00	13.18	31	**129**	1.44	6.90	0.00
2	10	15	**37**	**163**	**37**	182	2.11	0.00	11.66	39	**163**	1.91	5.41	0.00
2	15	10	**43**	273	**43**	302	7.27	0.00	10.62	45	**272**	5.23	4.65	−0.37
2	15	15	53	341	**52**	374	10.56	−1.89	9.68	54	**338**	8.44	1.89	−0.88
3	15	15	**25**	**160**	**25**	181	4.91	0.00	13.13	26	**160**	3.82	4.00	0.00
3	15	20	**31**	192	**31**	217	6.23	0.00	13.02	33	**191**	5.05	6.45	−0.52
3	20	15	33	275	**32**	308	14.44	−3.03	12.00	34	**273**	9.99	0.00	−0.73
3	20	20	43	356	**42**	394	19.14	−2.33	10.67	43	**352**	12.92	0.00	−1.12

Table 6
Results for the SSP-NPM-II set of instances.

m	n	l	ARV		$BRKGA_{Cmax}$					$BRKGA_{TFT}$				
			C_{max}	TFT	C_{max}	TFT	T	gap_{Cmax}(%)	gap_{TFT}(%)	C_{max}	TFT	T	$gap_{C_{max}}$(%)	gap_{TFT}(%)
4	40	60	325	4998	**304**	5393	727.62	−6.46	7.90	340	**4547**	187.68	4.62	−9.02
4	40	120	622	9672	**608**	10 279	2158.34	−2.25	6.28	664	**9221**	491.68	6.75	−4.66
4	60	60	509	11955	**446**	12 520	3397.35	−12.38	4.38	511	**10532**	492.28	0.39	−11.90
4	60	120	942	22620	**895**	23 780	3607.23	−4.99	5.13	980	**21190**	1278.12	4.03	−4.81
6	80	120	923	29299	**862**	31 854	3611.23	−6.61	8.72	976	**27656**	2330.22	5.74	−5.60
6	120	120	1468	68983	**1320**	74 541	3661.31	−10.08	8.06	1504	67517	3568.08	2.45	−2.13

solution was improved by 49.02%, and the average running time was 3.36 s. Among the local search methods, the job exchange contributed the most to the final quality of the solution, accounting for 45.26% of the improvements. Job insertion accounted for 39.40%, and 1-block grouping accounts for 15.14% of the improvements. The average standard deviation reported for all instances in this set was only 1.55, demonstrating the consistency of the proposed method. Again, the time limit stopping criterion was not triggered for this set of instances.

5.4.2. Comparison on SSP-NPM-II set

Table 6 presents the results for the SSP-NPM-II set, following the same standard as the previous table. Regarding the makespan, $BRKGA_{Cmax}$ reported new best average values for all groups of instances, with an average gap of −7.13%, varying between gap values of −2.25% and −12.38% for individual groups of instances. On average, $BRKGA_{Cmax}$ required 24.92 generations to improve the initial solution by 37.06%. Among the local search procedures, the job exchange contributed most to the final solution quality, accounting for 34.82% of improvements. Job insertion and 1-block grouping account for 33.42% and 31.76% of the improvements, respectively.

Considering all instances in this set, the reported standard deviation was only 2.67 (or 0.35%). The average running time was 2860.51 s. The maximum running time was reached in 65.60% of $BRKGA_{Cmax}$ individual runs, which was the main stopping criterion for this instance set. Given the large dimensions of the problems in this set, the maximum running time appears insufficient for $BRKGA_{Cmax}$ to run the 1000 predefined generations. However, the high quality of the solutions and the low number of generations required for convergence indicate that the running time is sufficient.

Regarding the total flow time, when compared with the state-of-the-art methods, $BRKGA_{TFT}$ reported new best average values for all group of instances, with a notable average gap of −6.36%, ranging between gap values of −11.90% and −2.13% for individual groups of instances. On average, the convergence of $BRKGA_{TFT}$ occurs after 562.33 generations, improving the initial solution by 40.04%. Among the improvements obtained by the local search procedures, the 1-block grouping contributes the most to the final quality of the solution and accounts for 65.55% of the improvements. Job insertion and job exchange accounted for 17.47% and 16.98% of the improvements, respectively.

Considering all instances in this set, the reported standard deviation was only 272.47 (or 1.14%). The average running time was 1391.51 s. The maximum running time was achieved in 23.06% of $BRKGA_{TFT}$ individual runs. As with $BRKGA_{Cmax}$, the defined maximum running time does not allow $BRKGA_{TFT}$ to process the predefined 1000 generations for the largest instances. However, it should be noted that this version requires significantly more generations than $BRKGA_{Cmax}$ for its convergence.

5.4.3. Statistical analysis

Statistical analyzes were performed to accurately compare the results obtained by the proposed methods with those reported by state-of-the-art methods. Among the proposed methods, only the results obtained by $BRKGA_{Cmax}$ were considered for the makespan analysis, and only those reported by $BRKGA_{TFT}$ were considered to analyze the total flow time. The average results for each group of instances were considered representative of the respective samples. First, we consider instances of the SSP-NPM-I set.

Concerning the makespan, the *Shapiro–Wilk* normality test (Shapiro and Wilk, 1965) confirmed the null hypothesis that the $BRKGA_{Cmax}$ ($W = 0.95634$, p-value $= 0.7746$) and ARV ($W = 0.95584$, p-value $= 0.7697$) 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 was a significant difference between the compared results. The t-test indicated that there was a significant difference and that $BRKGA_{Cmax}$ had the best average values ($t = -2.0494$, $df = 7$, p-value $= 0.0398$), with a significance level of 0.05.

Considering the total flow time, the Shapiro–Wilk normality test (Shapiro and Wilk, 1965) confirmed the null hypothesis that the $BRKGA_{TFT}$ ($W = 0.90532$, p-value $= 0.3223$) and ARV ($W = 0.9065$, p-value $= 0.3301$) results could be modeled according to a normal distribution with a confidence interval of 95%. Again, the parametric Student's t-test (Student, 1908) was applied to verify whether there was a significant difference between the compared results. The t-test indicated that there was a significant difference and that $BRKGA_{TFT}$ had the best average values ($t = -2.5825$, $df = 7$, $p - value = 0.01817$), with a significance level of 0.05.

A second statistical analysis was performed regarding the results generated for the instances of the SSP-NPM-II set. Considering the makespan, the *Shapiro–Wilk* normality test (Shapiro and Wilk, 1965) confirmed the null hypothesis that the $BRKGA_{Cmax}$ ($W = 0.96038$, p-value $= 0.8227$) and ARV ($W = 0.94688$, p-value $= 0.7149$) 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 was a significant difference between

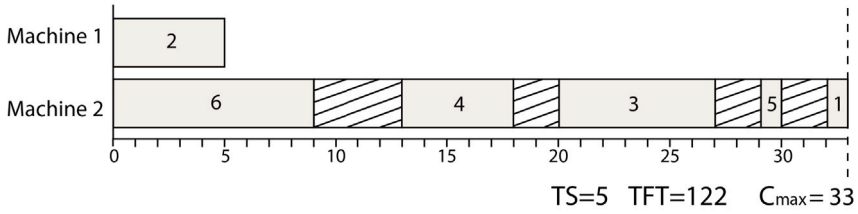


Fig. 8. Optimal solution for minimization of tool switches.

the compared results. The t-test indicated that there was a significant difference and that BRKGA$_{Cmax}$ had the best average values ($t = -3.0064$, $df = 5$, p-value = 0.01494), with a significance level of 0.05.

Considering the total flow time, the Shapiro–Wilk normality test (Shapiro and Wilk, 1965) rejected the null hypothesis that the BRKGA$_{TFT}$ ($W = 0.80698$, p-value = 0.06786) and ARV ($W = 0.81979$, p-value = 0.08788) results could be modeled according to a normal distribution with a confidence interval of 95%. Thus, the non-parametric *Wilcoxon signed-rank test* (Rey and Neuhäuser, 2011) was applied and confirmed that there was a significant difference between the compared methods and that BRKGA$_{TFT}$ had the best average values ($V = 0$, $p - value = 0.01776$), with a significance level of 0.05.

5.4.4. Discussion on objective functions

Although it is not the objective of this study to compare the results obtained by the different versions of the BRKGA, granted they were elaborated with different objectives, it is interesting to analyze the relationship between C_{max} and TFT. Tables 5 and 6 presented results for both BRKGA implementations concerning both objective functions.

Although $BRKGA_{Cmax}$ does not use the total flow time value to guide its evolutionary process, it presents an average gap of 11.74% for this objective function compared with the reference values for the SSP-NPM-I set. Conversely, BRKGA$_{TFT}$ does not use the makespan value to explore the search space, however, considering this objective function, it reported an average gap of only 3.66% compared with the reference values for the same set of instances.

Concerning the total flow time, when compared with the reference values for the SSP-NPM-II set, the BRKGA$_{Cmax}$ reported an average gap of 6.75%. Considering the makespan minimization, when compared with the reference values, the BRKGA$_{TFT}$ reported an average gap of 4.00%, with a minimum of 0.39% by instance group and a maximum of 6.75%. Although the objectives are different, in general, solutions achieved with the objective of minimizing the total flow time present better makespan values than the contrary.

The minimization of the number of tool switches (TS) is the third objective function considered in previous studies (Calmels, 2022; Cura, 2023). We decided not to include it in our study as we consider it a conflict with practical industrial settings with multiple machines. During exploratory experimentation, we confirmed our hypothesis that TS would result in a large number of jobs assigned to few machines. In fact, we analyzed each of the generated solutions and observed the same pattern in solution structure: a large number of jobs assigned to the machine with largest magazine capacity and small blocks of jobs without tool switches assigned to the other machines, causing unjustified large periods of idleness to all but few machines and prohibitively large production times.

Fig. 8 illustrates an optimal solution for TS, considering the instance presented in Table 1. To achieve the optimal solution with five-tool switches, Machine 1 processes only one job. In contrast, Machine 2 is overloaded, directly contributing to the large total flow time and makespan values.

TS is properly considered in makespan and total flowtime objective functions. It disregards in substantial amount the "multiple machines" dimension from the problem description, therefore, we consider the single machine SSP as the proper industrial setting for it. Given the abovementioned arguments and given that we could not find any argument in favor of TS in industrial settings with multiple machines, we decided respectfully not to include it in our manuscript.

6. Conclusion and future work

In this study, we proposed a new approach to the SSP-NPM, an $\mathcal{NP}$-hard problem with several practical applications in the industrial sector. The problem was approached by considering two distinct objective functions, i.e., the minimization of makespan and total flow time. For each objective function, we presented a version of the parallel metaheuristic BRKGA hybridized with VND. The versions are very similar and comprise a robust approach for the two objectives. The proposed method was analyzed using existing benchmark instances available in literature and compared with current state-of-the-art methods, including optimal solutions. Considering the reported results and statistical analyzes performed, we conclude that the proposed BRKGA outperforms the current state-of-the-art methods as it consistently improved or matched the best solution values for all instance groups for both objectives. Furthermore, all known optimal solutions for both objectives were matched. For the set containing the largest instances, the proposed method reported average gap values of −7.13% for the makespan and −6.36% for the total flow time. Despite its academic and practical relevance, the literature on SSP-NPM is limited. By continuing the study on the topic, creating cutting-edge approaches to the problem, and providing new and more competitive reference values for the available instances, we expect to stimulate future research, effectively expanding the specific literature and related works. Future work will focus on related scheduling problems and recently introduced different versions of the SSP-NPM, including, (i) the *sequence-dependent* version, in which the tool switching times vary according to the pair of removed and inserted tools; (ii) the *unsupervised* version, in which tool switches cannot occur during periods when human operators are not available; and (iii) the *tool sharing* version, which considers a single set of tools shared among unrelated machines. These problems can also be further extended to consider the practical aspect of tool switches originated by tool wearing.

CRediT authorship contribution statement

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

Data availability

The source code for the proposed methods is conveniently accessible online at https://github.com/larback/ssp-npm.

Acknowledgments

This work was supported by National Counsel of Technological and Scientific Development (Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq), Brazil and Universidade Federal de Ouro Preto, Brazil. This paper is dedicated in loving memory of Manny M. Carvalho, *fidus Achates*.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cor.2023.106509.

References

Allahverdi, A., 2015. The third comprehensive survey on scheduling problems with setup times/costs. European J. Oper. Res. 246, 345–378. http://dx.doi.org/10.1016/j.ejor.2015.04.004.

Allahverdi, A., Gupta, J., Aldowaisan, T., 1999. A review of scheduling research involving setup considerations. Omega 27, 219–239. http://dx.doi.org/10.1016/S0305-0483(98)00042-5.

Allahverdi, A., Ng, C., Cheng, T.E., Kovalyov, M.Y., 2008. A survey of scheduling problems with setup times or costs. European J. Oper. Res. 187, 985–1032. http://dx.doi.org/10.1016/j.ejor.2006.06.060.

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, 62–71. http://dx.doi.org/10.1016/j.ejor.2017.05.003.

Bard, J.F., 1988. A heuristic for minimizing the number of tool switches on a flexible machine. IIE Trans. 20, 382–391. http://dx.doi.org/10.1080/07408178808966195.

Bean, J.C., 1994. Genetic algorithms and random keys for sequencing and optimization. ORSA J. Comput. 6, 154–160. http://dx.doi.org/10.1287/ijoc.6.2.154.

Beezão, A.C., Cordeau, J.F., Laporte, G., Yanasse, H.H., 2017. Scheduling identical parallel machines with tooling constraints. European J. Oper. Res. 257, 834–844. http://dx.doi.org/10.1016/j.ejor.2016.08.008.

Calmels, D., 2019. The job sequencing and tool switching problem: state-of-the-art literature review, classification, and trends. Int. J. Prod. Res. 57, 5005–5025. http://dx.doi.org/10.1080/00207543.2018.1505057.

Calmels, D., 2022. An iterated local search procedure for the job sequencing and tool switching problem with non-identical parallel machines. European J. Oper. Res. 297, 66–85. http://dx.doi.org/10.1016/j.ejor.2021.05.005.

Calmels, D., Rajendran, C., Ziegler, H., 2019. Heuristics for solving the job sequencing and tool switching problem with non-identical parallel machines. In: Fortz, B., Labbé, M. (Eds.), Operations Research Proceedings 2018. Springer International Publishing, Cham, pp. 459–465. http://dx.doi.org/10.1007/978-3-030-18500-8_57.

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, 33–54. http://dx.doi.org/10.1007/BF01324874.

Cura, T., 2023. Hybridizing local searching with genetic algorithms for the job sequencing and tool switching problem with non-identical parallel machines. Expert Syst. Appl. 223, 119908. http://dx.doi.org/10.1016/j.eswa.2023.119908.

Fathi, Y., Barnette, K.W., 2002. Heuristic procedures for the parallel machine problem with tool switches. Int. J. Prod. Res. 40, 151–164. http://dx.doi.org/10.1080/00207540110076115.

Gökgür, B., Hnich, B., Özpeynirci, S., 2018. Parallel machine scheduling with tool loading: a constraint programming approach. Int. J. Prod. Res. 1–17. http://dx.doi.org/10.1080/00207543.2017.1421781.

Gonçalves, J.F., Resende, M.G., 2011. Biased random-key genetic algorithms for combinatorial optimization. J. Heuristics 17, 487–525. http://dx.doi.org/10.1007/s10732-010-9143-1.

González, M.A., Oddi, A., Rasconi, R., Varela, R., 2015. Scatter search with path relinking for the job shop with time lags and setup times. Comput. Oper. Res. 60, 37–54. http://dx.doi.org/10.1016/j.cor.2015.02.005.

Haddadi, S., Chenche, S., Cheraitia, M., Guessoum, F., 2015. Polynomial-time local-improvement algorithm for consecutive block minimization. Inform. Process. Lett. 115, 612–617. http://dx.doi.org/10.1016/j.ipl.2015.02.010.

Kummer, N.A.F., Buriol, L.S., de Araújo, O.C., 2020. 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. pp. 717–724. http://dx.doi.org/10.1145/3377930.3390209.

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. http://dx.doi.org/10.1016/j.orp.2016.09.002.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Comput. Oper. Res. 24, 1097–1100. http://dx.doi.org/10.1016/S0305-0548(97)00031-2.

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, 637–655. http://dx.doi.org/10.1016/j.ejor.2019.01.015.

OpenMP, 2019. The OpenMP api specification for parallel programming. https://www.openmp.org/. (Online; accessed 01 February 2023].

Özpeynirci, S., Gökgür, B., Hnich, B., 2016. Parallel machine scheduling with tool loading. Appl. Math. Model. 40, 5660–5671. http://dx.doi.org/10.1016/j.apm.2016.01.006.

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. http://dx.doi.org/10.1016/j.cor.2017.07.013.

Passmark, 2023. CPU benchmarks. http://www.cpubenchmark.net/cpu_list.php. (Accessed 25 February 2023).

Pinedo, M.L., 2012. Scheduling. Vol. 29. Springer, http://dx.doi.org/10.1007/978-3-319-26580-3.

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, 1140–1152. http://dx.doi.org/10.1016/j.ejor.2017.10.050.

Rey, D., Neuhäuser, M., 2011. Wilcoxon-Signed-Rank Test. Springer Berlin Heidelberg, Berlin, Heidelberg, http://dx.doi.org/10.1007/978-3-642-04898-2_616.

Shapiro, S.S., Wilk, M.B., 1965. An analysis of variance test for normality (complete samples). Biometrika 52, 591–611. http://dx.doi.org/10.1093/biomet/52.3-4.591.

Shirazi, R., Frizelle, G.D.M., 2001. Minimizing the number of tool switches on a flexible machine: an empirical study. Int. J. Prod. Res. 39, 3547–3560. http://dx.doi.org/10.1080/00207540110060888.

Soares, L.C.R., Carvalho, M.A.M., 2020. Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints. European J. Oper. Res. 285, 955–964. http://dx.doi.org/10.1016/j.ejor.2020.02.04.

Soares, L.C.R., Carvalho, M.A.M., 2022. Application of a hybrid evolutionary algorithm to resource-constrained parallel machine scheduling with setup times. Comput. Oper. Res. 139, 105637. http://dx.doi.org/10.1016/j.cor.2021.105637.

Spears, W.M., De Jong, K.D., 1995. On the Virtues of Parameterized Uniform Crossover. Technical Report, Naval Research Lab, Washington D.C, URL https://apps.dtic.mil/sti/pdfs/ADA293985.pdf.

Stecke, K.E., 1983. Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems. Manage. Sci. 29, 273–288. http://dx.doi.org/10.1287/mnsc.29.3.273.

Student, 1908. The probable error of a mean. Biometrika 1–25. http://dx.doi.org/10.1093/biomet/6.1.1.

Tang, C.S., Denardo, E.V., 1988. Models arising from a flexible manufacturing machine, part I: Minimization of the number of tool switches. Oper. Res. 36, 767–777. http://dx.doi.org/10.1287/opre.36.5.767.

Van Hop, N., Nagarur, N.N., 2004. The scheduling problem of pcbs for multiple non-identical parallel machines. European J. Oper. Res. 158, 577–594. http://dx.doi.org/10.1016/S0377-2217(03)00376-X.

Zeballos, L., 2010. A constraint programming approach to tool allocation and production scheduling in flexible manufacturing systems. Robot. Comput.-Integr. Manuf. 26, 725–743. http://dx.doi.org/10.1016/j.rcim.2010.04.005.